Title: Data7202 A4 Report

Name: Xinqian Wang

ID: 45654897

Tips: the code screenshot will be attached in the Appendix. For code script will also be attached in the folder.

1.

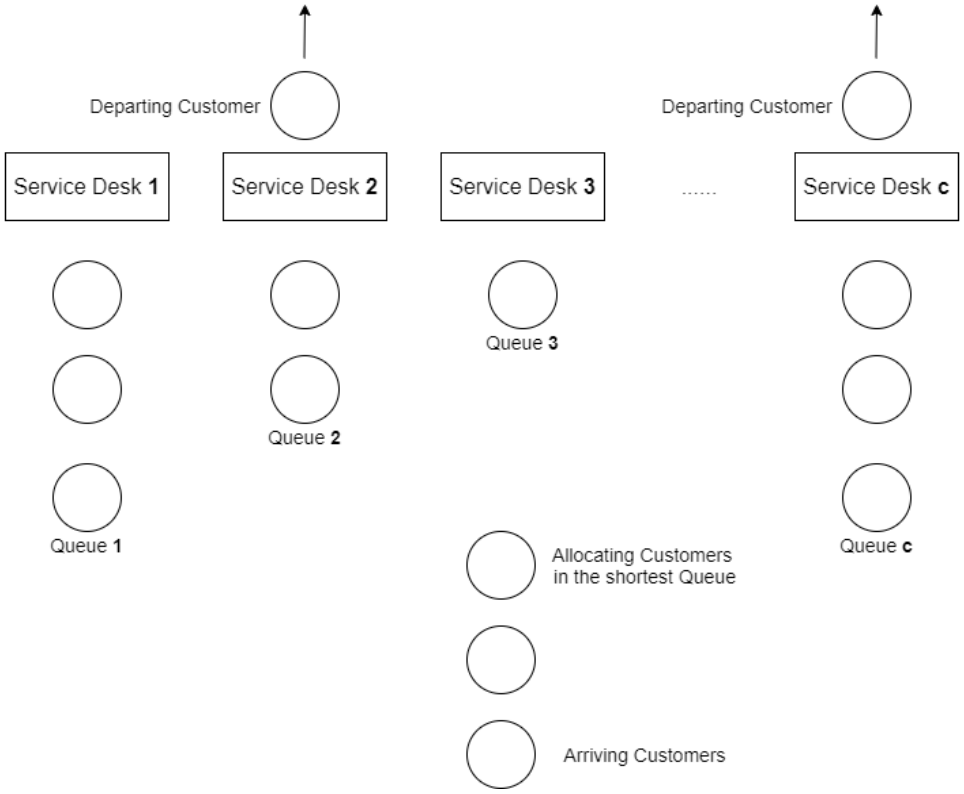
Question: Give the problem summary and describe the project objective.

Answer:

This is an M/M/C model, with given distribution of customers arrival(M) and their services length(M). There are C servers. The length of each queue can be infinite. The strategy for serving customers will be first come first service (FCFC).

Furthermore, we need to decide how many service desks (number of servers) to meet the requirement, which is in 90% of the customers do not exceed waiting time of 8 minutes. The criterion of whether we have met the requirement would be decided by a special rule. The rule is, in a fixed time (T=3000), we collection the data after the simulation and divided them into 50 batches for checking that the percentage of customers who's waiting time is below 8 for each batch. Such percentage should all above 90%. Additionally, we also need to use a Burn-In method, which is discarding the 30% data we collect finally. This process is trying to make sure the system can be trusted.

We also draw a figure to describe the simulation, as below:



- **2.**

Question: Give a specification of variables used in the simulation study. In addition, show a diagram that describes the project dynamics.

Answer:

The figure below are the state variables we made for customers:

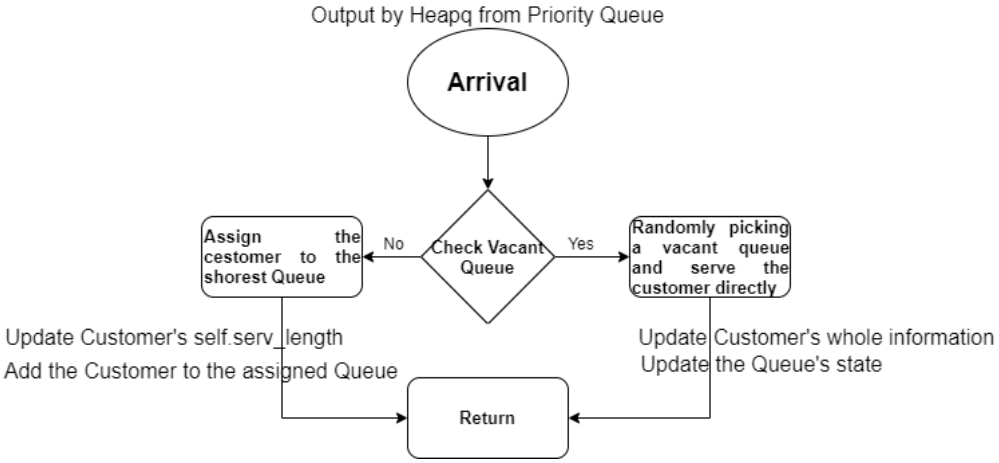
```
class SimData:
    '''
    SimData->Refer to the code provided by the Lab4_Data7202
    Contain the information for each customer
    '''
    def __init__(self, time_arrive, event_type, serv_1_start, serv_1_end, serv_length, serv_number):
        self.time_arrive = time_arrive
        self.event_type = event_type
        self.serv_1_start = serv_1_start
        self.serv_1_end = serv_1_end
        self.serv_length = serv_length
        self.serv_number = serv_number
```

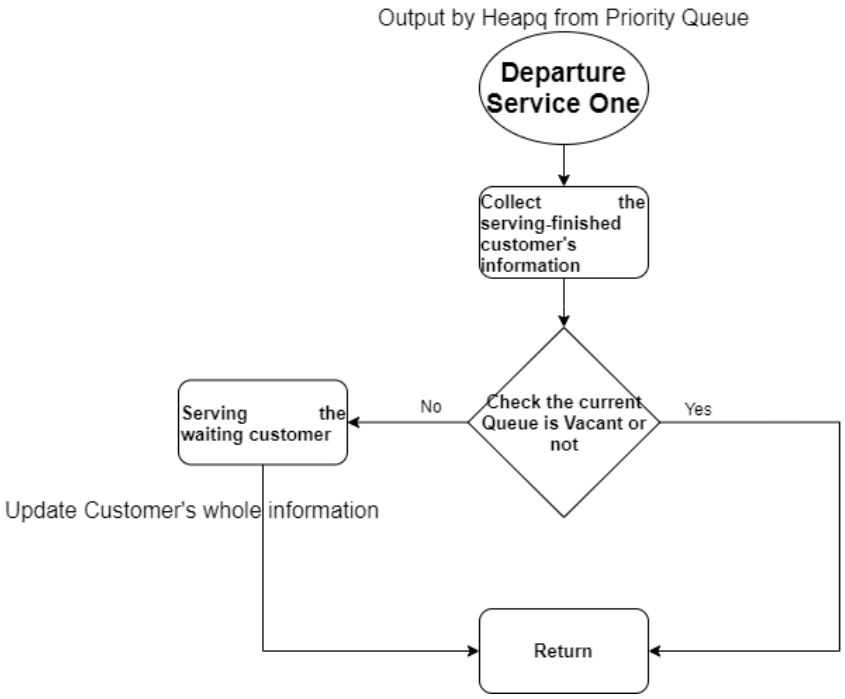
We considered adding customer's arriving time (**self.time_arrive**), customer's state type (self.event_type), customer's starting time of service 1 (self.serv_1_start), customer's ending time of service 1 (self.serv_1_end), customer's serving time length (self.serv_length), customer's queue index they are currently in(self.serv_number).

The variable used to save the whole information of customers is a Priority Queue. The Priority Queue is designed by comparing the **"self.time_arrive"** as the priority. The Priority Queue is realized by using a data structure called Heapq. The Heapq would self-madding a tree structure and store the customers' information. The function of Heapq will not be discussed here. We used a tuple as provided below. The Heapq would compare the first value and output the smallest, which is the first coming customer since it has the shortest arriving time (FCFC). However, the algorithm sometimes would throw one exception because there exists more than two customers which share the same arriving time. Considering such case, we add a random variable as the second parameter in the tuple and put them in the Priority Queue.

$$(\text{self.time\_arrive,random\_variable,customer information})$$

We also made two diagrams about the customers' work flow, which is Arrival State and Departure Service One. Additionally, we need to check the corresponding Queue when the customer came to the second state and serve the waiting customer in that Queue.



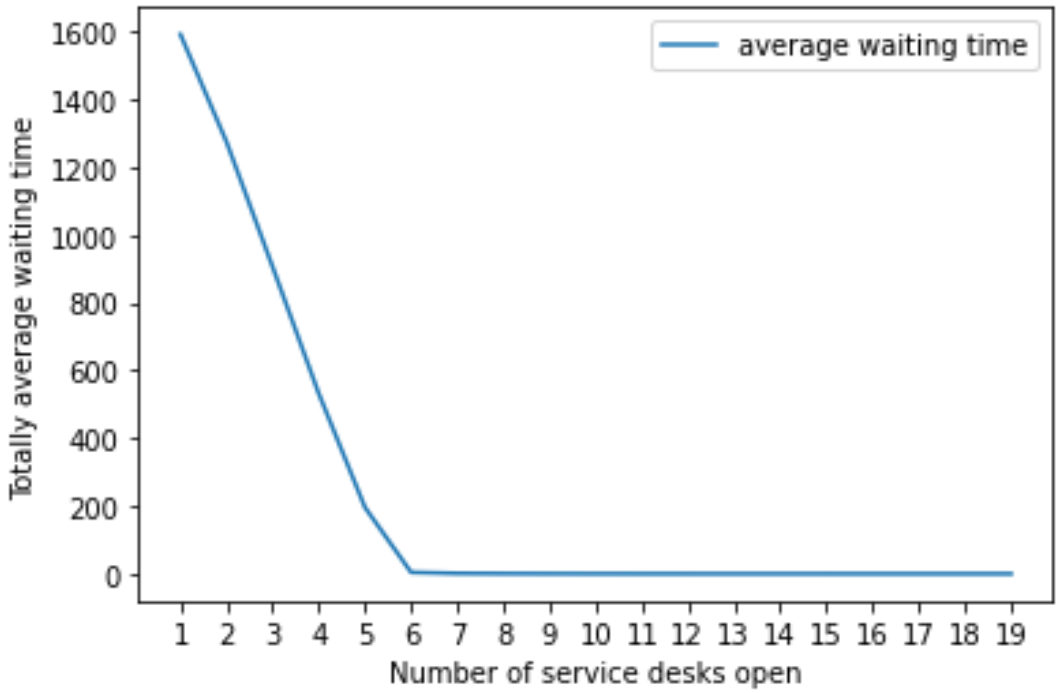


- **3.**

Question: Results and Analysis. Using tables and figures, present a clear outcome of your study. Present the corresponding confidence intervals.

Answer:

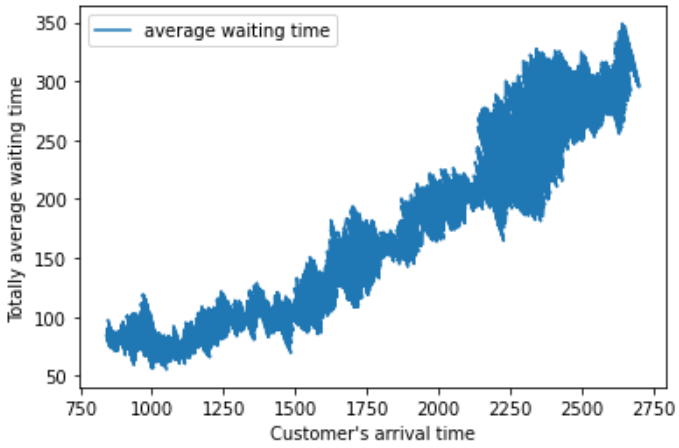
We first simulate the number of desks from 1 to 20, and made a line chart below:



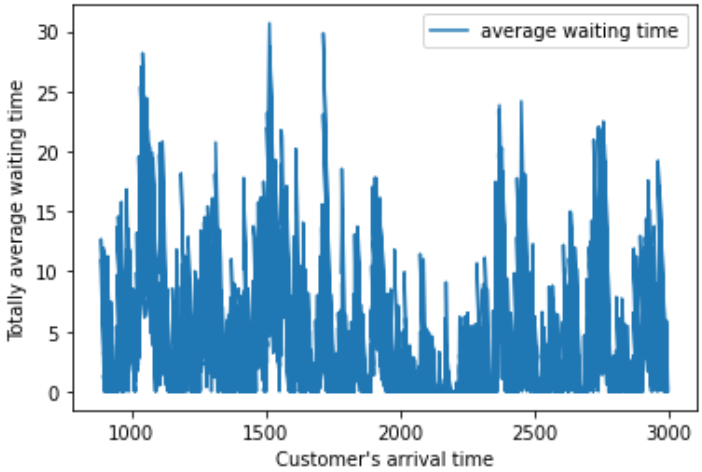
We can see from when table quantities greater than 5, the average waiting time came to decrease. As the result, we plot sever pictures with different service quantities.

The figures are shown as below.

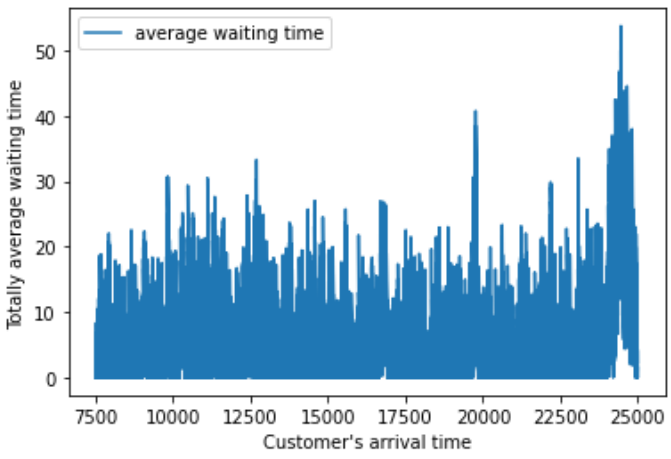
When we only open 5 service desks. The 95% CI for average waiting time in the system (214.22124983796888, 218.30722520960163). such value is insignificant since the average waiting time keeps increasing infinitely as the arrival time becomes longer.



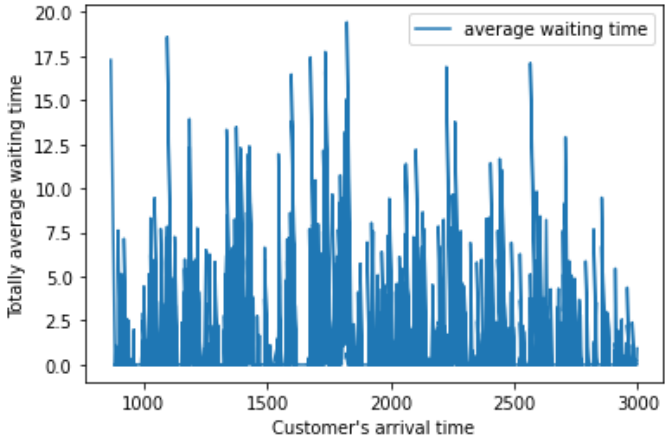
When we only open 6 service desks. The average waiting time seems to get controled and the 95% CI for average waiting time in the system (3.3156368067021713, 3.5761238880790183).



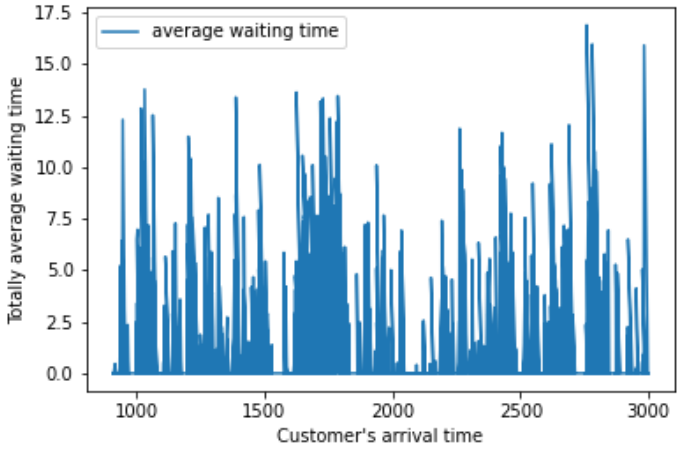
However, if increasing the time period T, from 3000 to 25000, but fixed to 6 service desks. We observe such figure as below and the 95% CI for average waiting time in the system (4.083336624812108, 4.199795985092057).



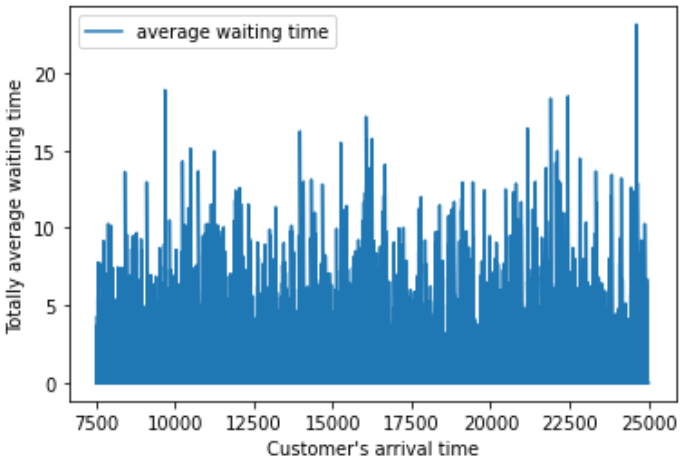
When we only open 7 service desks. The average waiting time seems to get controled and the 95% CI for average waiting time in the system (0.9086380565605148, 1.0410784729639124).



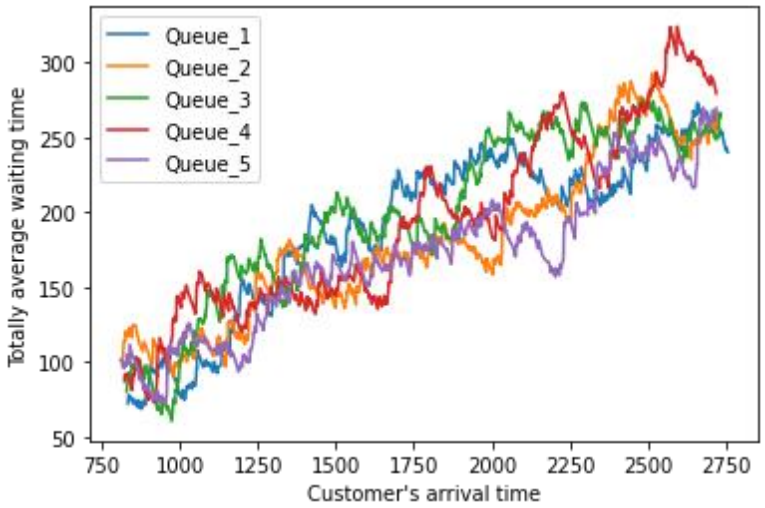
When we only open 8 service desks. The average waiting time seems to get controled and the 95% CI for total waiting time in the system (0.6125496125484537, 0.722308337805561).



If increasing the time period T, from 3000 to 25000, but fixed to 8 service desks. We observe such figure as below and the 95% CI for average waiting time in the system (0.39652378818849504, 0.42609887535505003). It is very impressive to observe such situation, when the time period increases to a huge amount. The 95% CI for total average waiting time is decreasing compared to the last figure. The figure is shown as below:

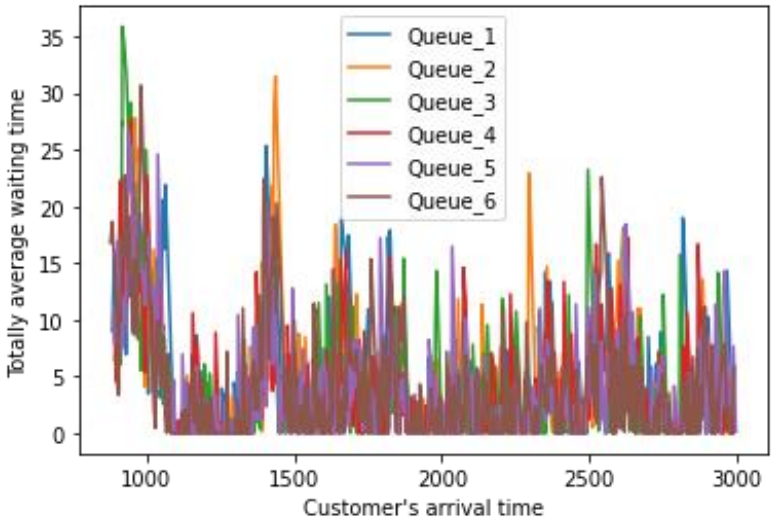


For comparation, we also plot another three figures with the quantities of servers of 5,6 and 7 (default time period of 3000). For each figure, we further the average waiting time for each queue. For the quantity of 5 service desks, the result is below:



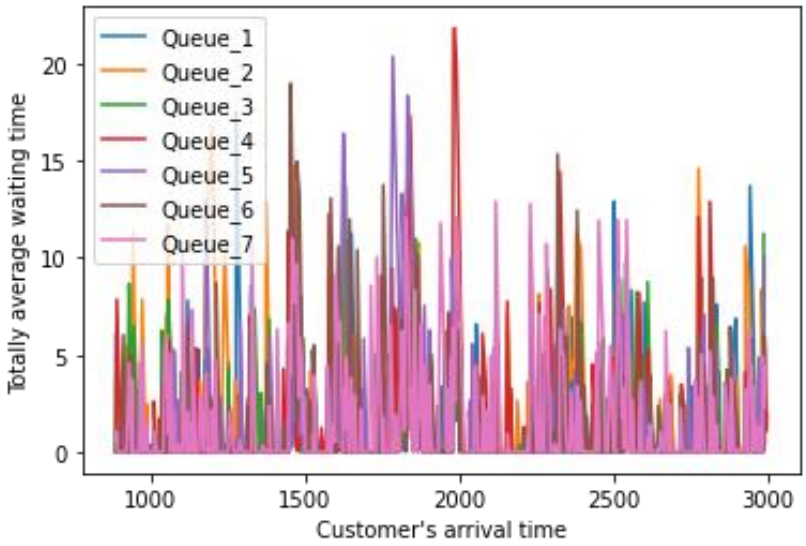
Queue Index	95% CI for average waiting time
1	(183.03281436892578, 191.245234592412)
2	(176.70201186690022, 184.05242055220603)
3	(191.55716705202562, 200.0593061565592)
4	(186.63800560808548, 195.5193203659782)
5	(167.26918785308723, 173.8516801976477)

For the quantity of 6 service desks, the result is below:



Queue Index	95% CI for average waiting time
1	(3.617233325480136, 4.406984910038794)
2	(3.3379144408102137, 4.101984696346521)
3	(3.2027252085598494, 3.9520650827090424)
4	(3.357352858683632, 4.076620692188868)
5	(3.1655212922210727, 3.8770115946999346)
6	(3.146247120789588, 3.842412368153278)

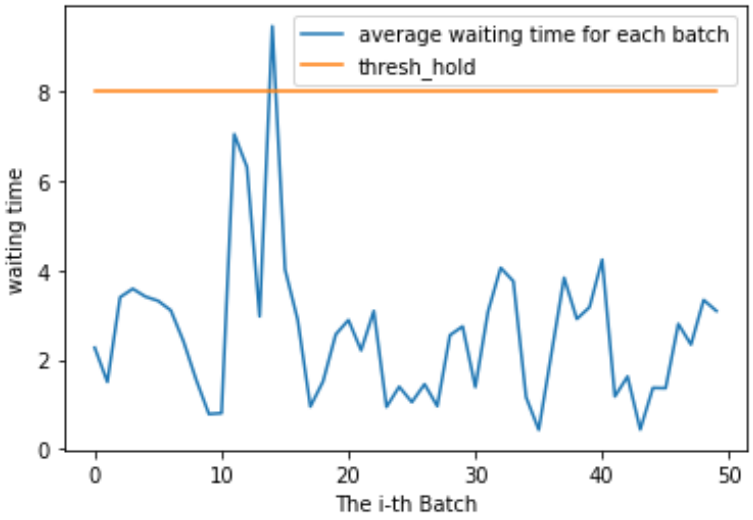
For the quantity of 7 service desks, the result is below:



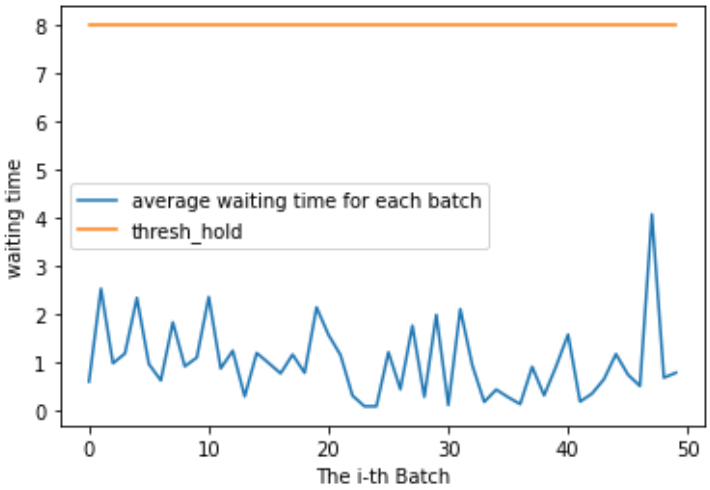
Queue Index	95% CI for average waiting time
1	(1.0323576102286456, 1.4015197884101278)
2	(1.242086262381764, 1.7096510083676617)
3	(0.9981394249033755, 1.341669697126188)
4	(1.1436553469076778, 1.614634478356628)
5	(1.1363498483109216, 1.5753044605916107)
6	(1.0009305389020056, 1.3898185118067725)
7	(1.0645734032616825, 1.461465980316747)

The calculation of the confidence interval will not take batch into considerations. From now, when we will divide the customer data we get after the Burn-In period into 50 batches to do further analyzing. Furthermore, from the research above, we would consider three value of service desks which is 6,7 and 8.

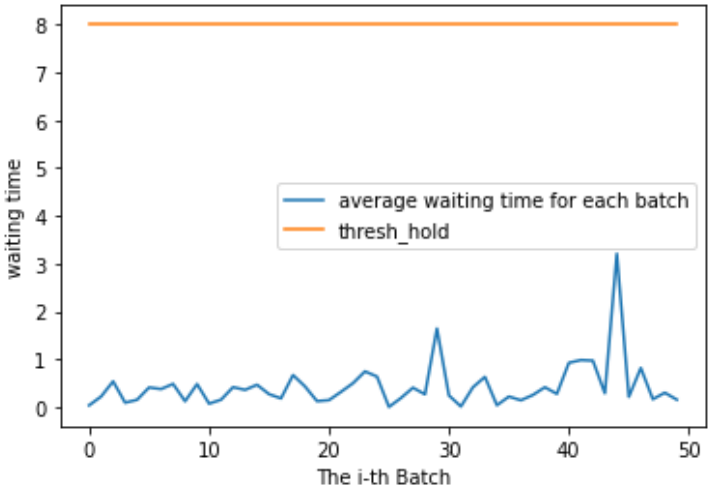
When number of service desks equals to 6. We use the data after the Burn-In period and divided the data into 50 batches to do research. We calculate the mean waiting time for each batches' customers and plot a figure as below:



When number of service desks equals to 7. We would observe the outcome as below:

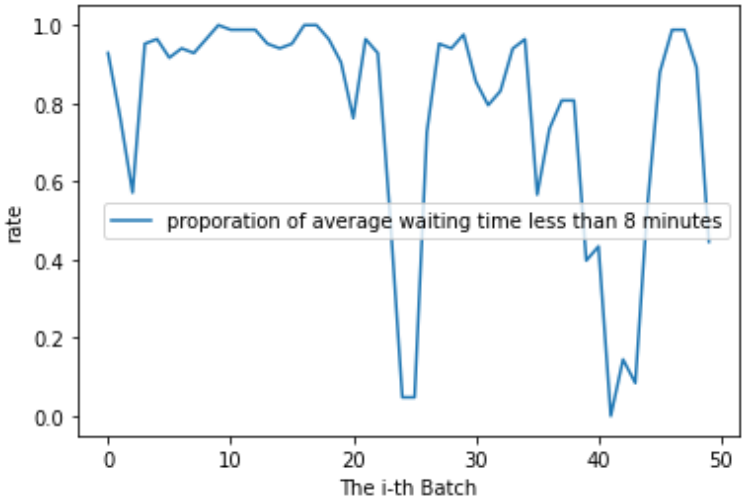


When number of service desks equals to 8. We would observe the outcome as below:

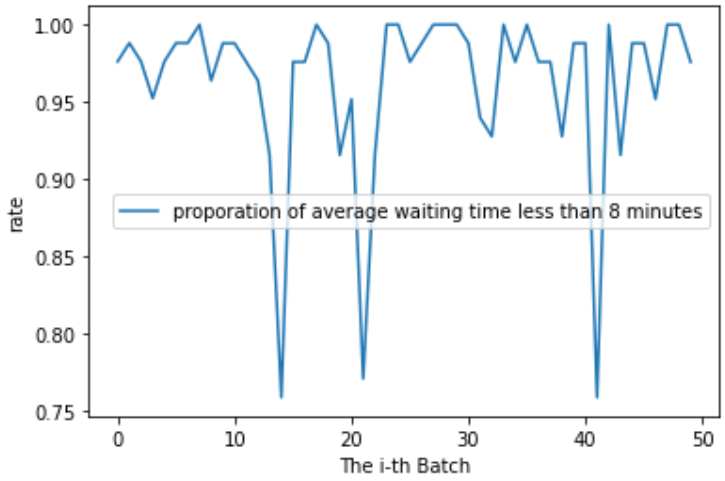


It seems with 7 desks the requirement would be satisfied. However, we still need to know the percentage of the customer in each batch which has wait longer than 8 minutes. So, we also plot some figures corresponding to the rate we just mentioned.

When number of service desks equals to 6. We would observe the outcome as below:

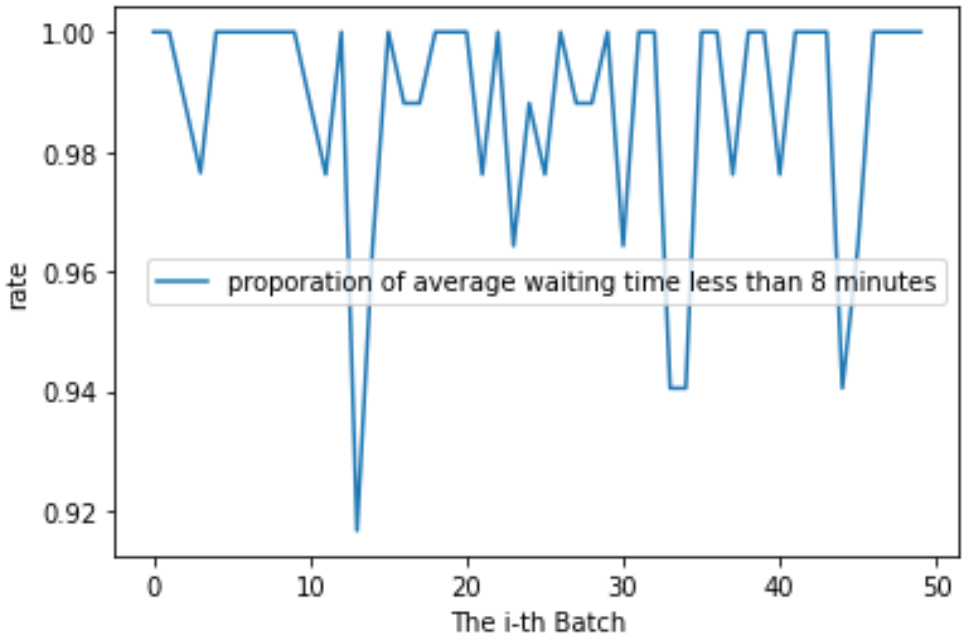


When number of service desks equals to 7. We would observe the outcome as below:



It seems desks number which equals to 7 would still can not guarantee the requirement. From the figure above.

We keep plotting, when number of service desks equals to 8. We would observe the outcome as below:



The requirement is met when the quantity of the service desk equals to 8.

We also made a table for the corresponding confidence intervals about all 50 batches. For each batch, there is an average waiting time, we use the 50-waiting time to check the 95% Confidence Intervals, from number of desks equals to 6,7 and 8.

Number of desks	95% CI average waiting time
6	(2.6984626583350675, 4.75811015391686)
7	(0.8496929362937202, 1.5243473387574695)
8	(0.33463480403905044, 0.5358908686165859)

- **4.**

Question: Formulate your conclusions.

Answer:

The minimum number of desks to satisfy the requirement, which is 90% of customers' waiting time do not exceed 8 minutes, is 8.

Additionally, I would like to talk about some check points when code the simulation. It would be useful when the researchers would try to make sure their simulation system is executing normally.

Test-1: Checking from a same queue, the i-th mission's end time should be equivalent to the (i+1)-th mission's starting time. Since there is no any description about the time intervals between these two missions.

Test-2: Checking the service length, from the service start to the end, the total interval should be equivalent to the customer's service length.

Test-3: Checking the customer's waiting time, which is the gap between when the customer arriving and the customer starting in the service.

Test-4: Check the state of the Customer, the data from the collection should be all with "Departure_Serve1", which means they all finish the task.

Test-5: Checking the length of the Queue.

- **5.**

Question: Appendix. Include all code files used. Explain their interaction and provide a clear and well-commented code.

Answer:

The code screen shot will be included in Appendix in the next page.

The basic logic of the code will be explained as here:

First, we initialize a customer, and put it in a Priority Queue through using Heapq. Then, the customer we put in will with all value with -1 and the default state of "Arrival". Then, the code detects the customer's state as "Arrival" and doing the following procedure. At the same time, it also will arrange the next arriving customer, by random sampling a row. In the following procedure, it will check the vacant Queues and randomly send them to the vacant Queue, or, in the case of all queues are fall, the customer will be sent to a shortest queue. In both situations, the customer's information about which queue he is sent to should be updated since the assumption is they will keep in the queue until served. Especially, In the first situation when the customer is sent to a vacant queue, the state of the queue will be changed from 0 to 1 and the state of the customer will be changed as "Departure_Serve1".

When the algorithm detects the customers' state as ""Departure_Serve1" in another iterations, the algorithm will record its information, saving it in a list. Then, the corresponding index of Queue state should be set as 0. Also, further check the current Queue's vacant or not and treat another customer if it is true.

Appendix

```python
import numpy as np
import heapq
import matplotlib.pyplot as plt
import pandas as pd
import random
import time
df = pd.read_csv("data.csv")
```

```python
class SimData:
    '''
    SimData->Refer to the code provided by the Lab4_Data7202
    Contain the information for each customer
    '''
    def __init__(self,time_arrive, event_type, serv_1_start,
serv_1_end,serv_length,serv_number):
        self.time_arrive = time_arrive
        self.event_type = event_type
        self.serv_1_start = serv_1_start
        self.serv_1_end = serv_1_end
        self.serv_length = serv_length
        self.serv_number = serv_number

    def Print(self):
        print(self.m_time_arrive," (", self.m_event_type,  ") : "
              ", ",self.m_serv_1_start, " - ", self.m_serv_1_end)
        print("service length:", self.m_serv_length)

def FindVacantDesk(State):
    '''
    Input State List
    Out put the index of the State List which state is 0
    '''
    Slist = []
    for s in range(len(State)):
        if State[s]<1:
            Slist.append(s)
    return Slist

def FindShortestDesk(Q):
    '''
    Input Queue
    Output a list of index refer to the elements in Queue which has the shortest length
    '''
    shortValue = 99999999
    shortList = []
    for q in Q:
        if len(q) < shortValue:
            shortValue = len(q)
    for i, q in enumerate(Q):
        if len(q) == shortValue:
            shortList.append(i)
    #return shortList
    random_queue = random.choice(shortList)
    return random_queue
```

```python
def simulation(n):
    #record list
    time_arr = []
    ellqx_len = []

    T = 3000 # simulation time
    ell = []
    priorityQueue = []
    Queue = []
    ServiceState = []
    N_Desk = n #Number of service desk to open
    for i in range(N_Desk):
        Queue.append([])
        ServiceState.append(0)


    # create first arrival
    arrival, service = df.sample(n=1).iloc[0]
    t_current = arrival
    # Initializing customer
    data = SimData(t_current,"ARRIVAL",-1,-1,service,-1)
    #put it in priorityQueue by using heapq
    heapq.heappush(priorityQueue,(t_current,random.random(),data))


    #start = time.time()
    while(t_current<T):
        #pop a object from priorityQueue which has the closest arrival time
        obj = heapq.heappop(priorityQueue)
        t_current = obj[0]
        event = obj[2]

        # record queus length
        time_arr.append(t_current)
        q_length = []
        for i in range(N_Desk):
            q_length.append(len(Queue[i]))
        ellqx_len.append(q_length)

        if(event.event_type == "ARRIVAL" and event.serv_number<0):
            # handle arrival
            # schedule the next arrival
            next_arrival, next_service = df.sample(n=1).iloc[0]
            t_next = t_current + next_arrival
            data = SimData(t_next,"ARRIVAL",-1,-1,next_service,-1)
            heapq.heappush(priorityQueue,(t_next,random.random(),data))

            vacant_list = FindVacantDesk(ServiceState)
            if len(vacant_list)>0.9:
                #random select a random_queue
                random_queue = random.choice(vacant_list) #Index of the Queue
                #Update the chosen queue
                ServiceState[random_queue] = 1
                #Update the start time, end time, customer state, the number of Queue the
customer in
                event.serv_1_start = t_current
                event.serv_1_end = t_current + event.serv_length
                event.event_type = "DEPARTURE_SERV1"
                event.serv_number = random_queue
                #Add it into priorityQueue
                heapq.heappush(priorityQueue,(event.serv_1_end,random.random(),event))
            else:
                # We asign to the smallest list
                random_queue = FindShortestDesk(Queue) #Index of the Queue
                #Add the customer to the corresponding Queue
                event.serv_number = random_queue
                Queue[random_queue].append(event)
            continue

        if(event.event_type == "DEPARTURE_SERV1"):
            ell.append(event)
            # Set the corresponding queue to be vacant
            ServiceState[event.serv_number] = 0
            if len(Queue[event.serv_number])!=0:
                obj_wait_inq1 = Queue[event.serv_number].pop(0)
                obj_wait_inq1.serv_1_start = t_current
                obj_wait_inq1.event_type = "DEPARTURE_SERV1"
                obj_wait_inq1.serv_1_end = t_current + obj_wait_inq1.serv_length
                heapq.heappush(priorityQueue
,(obj_wait_inq1.serv_1_end,random.random(),obj_wait_inq1))
                ServiceState[obj_wait_inq1.serv_number] = 1
            continue
    #end = time.time()
    #print('executing time: ', end-start)


    return ell,time_arr,ellqx_len
```

```python
n = 8
ell,_,_ = simulation(n)

# Calculate the average waiting time in the system
BurnIn = int(len(ell)*0.3)
ell_w_time = np.zeros(len(ell)-BurnIn)

for i in range(0,len(ell)-BurnIn):
    event = ell[i+BurnIn]
    wait_t = event.serv_1_start - event.time_arrive
    ell_w_time[i] = wait_t

Batches = np.array_split(ell_w_time, 50)
B_Mean = [np.mean(b) for b in Batches]
B_List = [i for i in range(len(Batches))]
threshold_List = [8 for i in range(len(Batches))]

#tmean = np.mean(ell_w_time)

tmean = np.mean(B_Mean)
tstd = np.std(B_Mean)/np.sqrt(len(B_Mean))
print("95% CI for total waiting time in the system (", tmean - 1.96*tstd, " , ", tmean + 1.96*tstd, ")")


plt.plot( B_List,B_Mean, label="average waiting time for each batch")
plt.plot( B_List,threshold_List, label="thresh_hold")
plt.xlabel("The i-th Batch")
plt.ylabel("waiting time")
plt.legend()
```

```python
Wait = []
for b in Batches:
    n = len(b)
    count = len(['yes' for i in b if i< 8])
    Wait.append(count/n)

plt.plot( B_List,Wait, label="proporation of average waiting time less than 8 minutes")
plt.xlabel("The i-th Batch")
plt.ylabel("rate")
plt.legend()
```

```python
K = []
for n in range(1,20):
    ell,time_arr,ellqx_len = simulation(n)
    BurnIn = int(len(ell)*0.3)
    ell_w_time = np.zeros(len(ell)-BurnIn)
    ell_arr_time = np.zeros(len(ell)-BurnIn)

    for i in range(0,len(ell)-BurnIn):
        event = ell[i+BurnIn]
        wait_t = event.serv_1_start - event.time_arrive
        ell_w_time[i] = wait_t

    tmean = np.mean(ell_w_time)
    K.append(tmean)

plt.plot([str(i) for i in range(1,20)], K,label="average waiting time")
plt.xlabel('Number of service desks open')
plt.ylabel('Totally average waiting time')
plt.legend()
```

```
n = 8
ell,time_arr,ellqx_len = simulation(n)


# Calculate the average waiting time in the system
BurnIn = int(len(ell)*0.3)
ell_w_time = np.zeros(len(ell)-BurnIn)
ell_arr_time = np.zeros(len(ell)-BurnIn)

for i in range(0,len(ell)-BurnIn):
    event = ell[i+BurnIn]
    wait_t = event.serv_1_start - event.time_arrive
    ell_w_time[i] = wait_t
    ell_arr_time[i] = event.time_arrive

tmean = np.mean(ell_w_time)
tstd = np.std(ell_w_time)/np.sqrt(len(ell_w_time))

print("95% CI for total waiting time in the system (", tmean - 1.96*tstd, " , ", tmean + 1.96*tstd, ")")

plt.plot(ell_arr_time,ell_w_time, label="average waiting time")
plt.xlabel("Customer's arrival time")
plt.ylabel('Totally average waiting time')
plt.legend()
```

```
BurnIn = int(len(ell)*0.3)
for n in range(n):
    ell_arr_time = []
    ell_w_time = []
    name = "Queue_"+str(n+1)
    for i in range(BurnIn,len(ell)):
        event = ell[i]
        if event.serv_number == n:
            ell_arr_time.append(event.time_arrive)
            ell_w_time.append(event.serv_1_start-event.time_arrive)

    tmean = np.mean(ell_w_time)
    tstd = np.std(ell_w_time)/np.sqrt(len(ell_w_time))
    print("for ",name,"95% CI for waiting time in the system (", tmean - 1.96*tstd, " , ", tmean + 1.96*tstd, ")")
    plt.xlabel("Customer's arrival time")
    plt.ylabel('Totally average waiting time')
    plt.plot(ell_arr_time,ell_w_time, label=name)
plt.legend()
```

Thank you for reading my report

I look forward you to running my code.