STA240 Final Project

Anthony Zhao, Abby Li, William Yan

Scenario 1

Customer Arrival

Poisson process (rate = λ)

- T_k: Arrival time of the kth customer
- W_k: Time between the $k-1$th arrival and the kth arrival

$$W_k = T_k - T_{k-1}.$$

$W_k \sim Pois(\lambda)$

where $\lambda = 5$ customers per hour

Service Time

$S_k \sim Exp(\lambda)$

where $\lambda = 6$ customers per hour, so the average customer needs to wait 1/6 hours = 10 minutes.

Arrival Times

```r
library(tidyverse)

# simulating the arrival times of customers throughout the day

# Poisson process (lambda = 5)
# Tk= arrival time of the kth customer
# Wk= time between the k-1th customer arrival and the kth customer arrival where Wk ~ Pois(la

# set parameter
lambdaA <- 5 # in units: customers per hour
opening_time <- hm("10:00")
closing_time <- hm("22:00")
hours <- hour(closing_time) - hour(opening_time) # operating hours: 10am to 10pm
total_time <- hours*60 # operating hours in minutes
lambdaA <- lambdaA/60 # customers per minute
# converting to minutes because our lambda is low, and we can can get greater precision in ar

n <- ceiling(lambdaA*total_time) # max number of customers the store can have throughout the

# generate W1,..,Wn (calculating the time between the arrival times of 2 customers)
W_sample <- rexp(n, rate= lambdaA)

# calculate T or the arrival times by summing together the Wi arrival times

T_sample <- numeric(n)

for(i in 1:n) {
  T_sample[i] <- sum(W_sample[1:i])
}

# all possible arrival times of customers throughout the day (X minutes after opening)

# however, the store is only open for 12 hours or 720 minutes so we must get rid of the valu

arrival_times <- T_sample[T_sample <= total_time]

arrival_times
```

```
 [1]  20.21797  21.64741  26.30426  33.05791  35.95708  54.29261  56.50035
 [8]  63.22275  71.55870  73.10753  85.64743  89.64943 110.67433 114.01342
[15] 123.16846 128.59290 145.94538 150.25641 187.60263 195.64777 199.45562
[22] 214.25703 236.22032 246.21430 247.25599 259.52325 287.27002 295.22236
```

```
[29] 302.06511 350.36008 368.03409 373.80760 376.81845 382.24956 402.58220
[36] 407.10850 408.10209 420.57042 441.73462 455.31487 463.69774 465.79580
[43] 476.64750 509.26662 524.30741 541.20484 549.98488 553.51512 555.61186
[50] 558.05552 591.85063 603.39623 605.08350 612.79805 653.88995 662.53698
[57] 671.13595 673.04601 689.16875 701.85339
```

```r
opening_time + minutes(floor(max(arrival_times)))
```

```
[1] "10H 701M 0S"
```

Arrival Times Analysis

In this simulation, the number of customers that will be arriving within the operating hours is 60, with the first customer arriving 20 minutes after opening and the last customer arriving 19 minutes before closing

Serving Times

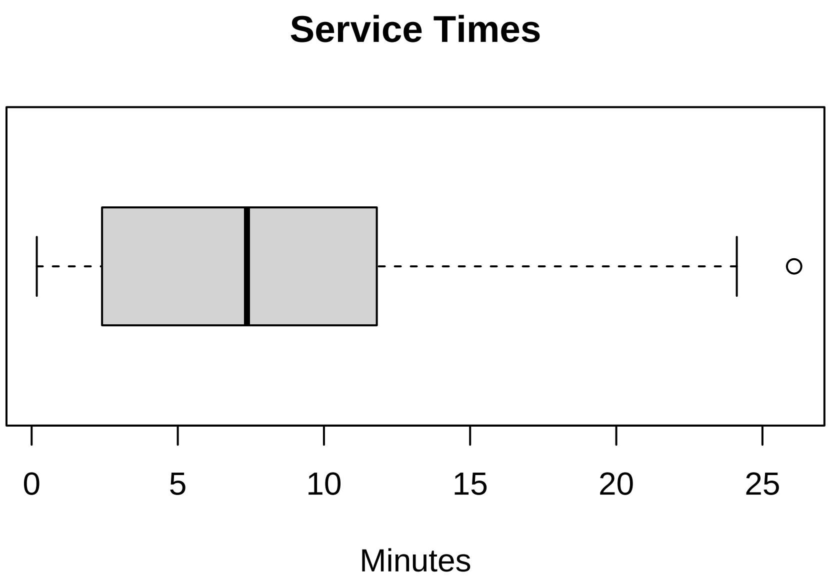
```r
# given the output from above, simulate the serving times of customers before they leave

# notice that service time is modeled by exp(6)
lambdaS <- 6 # customers per hour
lambdaS <- lambdaS/60 # customers per minute

# simulate customer's service time
# n= only simulating the service time for those where T_sample <= total_time
service_times <- rexp(length(arrival_times), rate= lambdaS)

#these are the serving times for each arriving customer before they leave
service_times
```

```
 [1]  8.1158040  0.2797043  4.2201281  5.8182636 16.3785962 12.3563612
 [7]  3.5586315  5.9784942 14.3580463  0.9092807 26.0833609  0.1764882
[13] 18.7746141 14.9228047  7.2513795 20.6061880 11.2603162  3.5441898
[19] 15.6693408  2.2340941  1.7176359  1.4846389  6.0688846 24.1217204
[25]  6.5945830  2.3368849  1.3631487  2.0045298 11.0230319  6.8629959
[31]  4.0095848  7.4816581  9.3394115  1.7739955 17.1579428  0.2480081
[37]  8.5917537  8.7764249  9.0687938  9.5111496  5.9191016  8.1866952
[43]  1.7219185  4.6648106  9.1558326 13.3916202  0.6292991  4.3944412
```

```
[49] 14.8255393  2.4831620  1.4818847 13.0078279  1.7709570  7.7765662
[55] 10.7259999 10.3160840  7.1063127 11.0411409 17.8741058 18.0040994
```

Serving Times Analysis

```r
boxplot(service_times, horizontal= TRUE, main= "Service Times", xlab= "Minutes")
```



The average service time is 8 minutes, with the data skewed right, consistent with an exponential distribution. This indicates that service times tend to lower.

Waiting Times

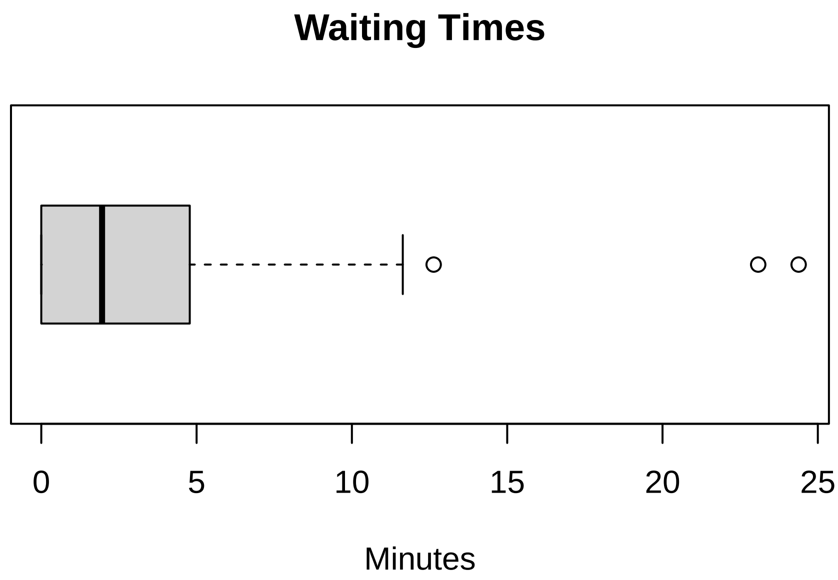
```r
# determining waiting times

# for each observation (customer), calculate when the service begins and when it ends
# serving ends = service begins + service time
# service begins: either when the customer walks in, or when the previous customer leaves (as

# compare this to the arrival time
# if arrival time > time service ends then wait time = 0
```

```
# but if arrival time < service time ends then wait time = time service ends- arrival time

# variable initialization
waiting_times <- numeric(length(arrival_times))  # generating times for each customer
service_start <- numeric(length(arrival_times))
service_end <- numeric(length(arrival_times))
current_end <- numeric(0) # service end time for current customer (i)

# iterate over each customer
for (i in 1:length(arrival_times)) {

  # only includes observations where service time > arrival time => which means there is a wa
  # gets rid of observations where service < arrival time => 0 wait time
  if (length(current_end) > 0) {
    current_end <- current_end[current_end > arrival_times[i]]
  }

 if (length(current_end) == 0) {
   # scenario 1: if there is no waiting time, service starts at the customer arrival
    service_start[i] <- arrival_times[i]
  } else {
    # scenario 2: if there is a waiting time, service starts at the end of the previous custo
    service_start[i] <- min(current_end)
  }

  # update the service end time for current customer by adding when service starts and how lo
  service_end[i] <- service_start[i] + service_times[i]

  # add this service end time to current end services
  current_end <- c(current_end , service_end[i])

  # update waiting time
  waiting_times[i] <- service_start[i] - arrival_times[i]
}


simulation_results <- data.frame(
  customer = 1:length(arrival_times),
  arrival_time = arrival_times,
  service_length = service_times,
  service_start = service_start,
  service_end = service_end,
```

```r
  waiting_time = waiting_times
)

print(head(simulation_results, 15)) # printing first 15 customers
```

```
   customer arrival_time service_length service_start service_end waiting_time
1         1     20.21797      8.1158040      20.21797    28.33377   0.00000000
2         2     21.64741      0.2797043      28.33377    28.61348   6.68636271
3         3     26.30426      4.2201281      28.33377    32.55390   2.02951011
4         4     33.05791      5.8182636      33.05791    38.87617   0.00000000
5         5     35.95708     16.3785962      38.87617    55.25476   2.91908902
6         6     54.29261     12.3563612      55.25476    67.61113   0.96215451
7         7     56.50035      3.5586315      67.61113    71.16976  11.11077772
8         8     63.22275      5.9784942      67.61113    73.58962   4.38837519
9         9     71.55870     14.3580463      73.58962    87.94767   2.03092293
10       10     73.10753      0.9092807      73.58962    74.49890   0.48208630
11       11     85.64743     26.0833609      87.94767   114.03103   2.30023372
12       12     89.64943      0.1764882     114.03103   114.20752  24.38160165
13       13    110.67433     18.7746141     114.03103   132.80564   3.35670176
14       14    114.01342     14.9228047     114.03103   128.95383   0.01761204
15       15    123.16846      7.2513795     128.95383   136.20521   5.78537115
```

```r
boxplot(waiting_times, horizontal= TRUE, main= "Waiting Times", xlab= "Minutes")
```




```
mean(waiting_times)
```

```
[1] 3.624185
```

Waiting times tend to be short, if not zero, and on average, the waiting time is on average 4 minutes.

Scenario 2

Arrival and Service

Assumptions:

1. 5 dining tables and L chefs with operating hours 10am - 10pm
2. each table only seats one customer
3. service time modeled by an exponential distribution with rate $S = 3L$, so that the more chefs there are, the faster the service times become **(this is not very realistic)**

```r
# first, we generate the arrival times similar in scenario 1
lambdaA <- 24 # per hour
opening_time <- hm("10:00")
closing_time <- hm("22:00")
hours <- hour(closing_time) - hour(opening_time)
total_time <- hours*60 # operating hours in minutes
lambdaA <- lambdaA/60 # per minute

n <- ceiling(lambdaA*total_time) # max number of customers
W_sample <- rexp(n, rate= lambdaA)
T_sample <- numeric(n)

for(i in 1:n) {
  T_sample[i] <- sum(W_sample[1:i])
}

arrival_times <- T_sample[T_sample <= total_time]

# next, we generate the service times similar to scenario 1
# make a function to do this
calc_service_times <- function(arrivals, chefs) {
  # Ensure rate is per unit time
  minute_rate = (3*chefs) / 60
  services = rexp(length(arrivals), rate = minute_rate)
  return(services) # in minutes
}
# if we only have one chef
service_times <- calc_service_times(arrivals = arrival_times, chefs = 2)
```

Waiting Times

To model waiting times, we iterate through the day minute by minute.

```r
tables <- 5
arrival_times_temp <- arrival_times

# number of people in line each minute
queue_size_history <- numeric(total_time)

# number of tables occupied each minute
occupied_tables_history <- rep(0, total_time)
```

```r
# timer to track remaining waiting time for each table in the restaurant
# each element is one table in the restaurant
# -1 means empty
# otherwise, number of remaining service minutes
tables_timer <- rep(-1, tables)

# the amount of minutes each customer of that day waited
waiting_times <- numeric(0)

# the arrival_times indices of the people currently in line
# in order to know how long their eventual service time will be
queue <- numeric(0)

# an internal counter separate from the time
customers_entered <- 0
for (i in 1:total_time) {
  occupied_tables_history[i+1] = occupied_tables_history[i]

  # update the waiting timer for all occupied tables
  tables_timer[tables_timer > 0] <- tables_timer[tables_timer > 0] - 1
  # update the number of available tables in the next minute
  # based on the number of tables who have finished timers
  occupied_tables_history[i+1] = occupied_tables_history[i+1] - sum(tables_timer == 0)
  # mark the finished tables as available tables for the next minute
  tables_timer[tables_timer == 0] <- tables_timer[tables_timer == 0] - 1

  # has the next customer arrived?
  if(length(arrival_times_temp) > 0){
    if(arrival_times_temp[1] < i) {
      # if so, add them to the back of the queue
      queue = c(queue, as.integer(customers_entered+1)) # add 1 for 1-indexing
      # remove the 1st element of arrival_times
      arrival_times_temp = arrival_times_temp[-1]
      # start the waiting timer for this customer by appending 0
      waiting_times = c(waiting_times, 0)

      customers_entered = customers_entered + 1
    }
  }
  # are any tables currently open and there is a person in line?
  if(occupied_tables_history[i+1] < tables & length(queue) > 0) {
    # if so, then seat the first person in line
```

```r
    # at the first available table
    for (j in 1:tables) {
      if(tables_timer[j] == -1) {
        # queue[1] has the customer index of the first person in line
        tables_timer[j] = round(service_times[queue[1]])
        break
      }
    }
    # the next minute there will be one more occupied table
    occupied_tables_history[i+1] = occupied_tables_history[i+1] + 1
    # remove the first person in the queue
    queue = queue[-1]
  }
  # update the waiting time for each person in the queue
  for (customer_index in queue) {
    waiting_times[customer_index] = waiting_times[customer_index] + 1
  }
  # keep track of how long the line is at each minute
  queue_size_history[i] = length(queue)
}

queue_size_history <- queue_size_history[-1]
occupied_tables_history <- occupied_tables_history[-1]

queue_size_history
```

```
  [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
 [26] 1 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0
 [51] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [76] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[101] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[126] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[151] 0 0 0 0 0 0 0 0 0 0 0 0 1 2 2 3 3 3 2 2 2 2 2 1 0
[176] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[201] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[226] 0 0 0 0 0 0 0 1 1 1 1 2 3 2 2 1 2 2 1 0 1 0 0 0 0
[251] 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 1 1 2 2 2
[276] 1 2 2 1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[301] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[326] 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[351] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 1 1 2
[376] 2 2 1 0 0 0 0 0 1 1 1 1 2 2 2 2 1 2 2 2 2 3 4 4 3
```

```
[401]  3  3  2  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
[426]  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1
[451]  0  0  1  1  1  2  3  4  4  4  4  4  3  2  3  4  3  4  4  4  4  3  3  3  4
[476]  5  4  4  4  3  3  3  3  4  4  4  5  5  5  6  6  6  7  7  8  9 10 11 12 12
[501] 13 14 14 14 13 14 15 15 15 15 15 16 16 16 17 17 17 17 16 16 17 16 16 16 17
[526] 17 17 18 18 18 18 18 18 19 20 21 22 23 24 24 24 24 24 24 23 22 21 21 20 20
[551] 20 19 20 20 20 21 21 21 21 22 22 22 21 21 22 22 21 20 21 21 20 19 18 17 17
[576] 16 16 15 16 16 17 17 17 17 17 17 18 18 18 19 18 17 16 16 15 14 13 12 11 11
[601] 12 13 13 14 13 13 14 14 15 15 14 13 14 14 15 14 14 15 16 16 16 17 18 18 18
[626] 18 17 16 16 15 15 16 17 17 17 17 17 16 15 15 15 15 15 14 15 16 16 16 16 16
[651] 16 16 17 17 17 17 17 17 18 17 17 18 18 17 17 16 15 16 17 18 19 19 18 18 18
[676] 18 18 17 16 16 15 14 14 13 13 13 13 13 13 13 12 11 11 11 11 11 10 10  9  9
[701]  9  9  8  8  8  7  7  6  5  5  5  5  4  4  3  3  3  3  2
```

```
occupied_tables_history
```

```
  [1] 1 2 2 2 2 2 3 3 3 3 3 2 3 2 3 4 5 5 5 5 5 5 4 4 5 5 5 5 5 5 5 5 5 5 5 5 5
 [38] 5 5 5 5 5 5 5 5 5 5 4 4 5 5 5 4 4 3 3 3 4 5 5 5 4 4 5 5 4 4 3 3 4 5 5 4 3
 [75] 3 3 3 2 2 3 2 1 1 1 1 2 1 2 3 3 2 2 3 4 4 4 4 3 4 4 4 3 4 5 4 4 4 4 3 4 3
[112] 3 4 3 3 3 2 2 3 4 4 3 4 3 3 3 4 3 3 2 2 2 2 2 2 3 3 3 3 3 3 2 3 3 2 2 2 2
[149] 3 3 2 2 3 3 3 4 4 4 5 5 5 5 5 5 5 5 5 5 5 4 5 5 4 5 5 4 4 2 2 1 2 2 2 3 1
[186] 1 1 2 1 1 1 1 2 2 3 3 3 3 4 4 4 3 3 3 2 2 2 1 1 1 1 1 1 2 2 2 1 1 2 2 2 2
[223] 2 3 3 3 3 4 3 4 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 3 3 3 3 4 5
[260] 5 4 3 2 3 4 4 4 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 4 4 5 5 5 5 3
[297] 3 3 2 2 2 2 2 3 3 3 3 3 4 4 3 4 4 4 4 2 3 2 2 2 2 2 2 2 3 4 5 5 5 5 5 5 4
[334] 4 3 3 2 3 3 3 3 1 1 2 2 2 3 3 3 4 4 3 3 4 3 3 4 4 4 4 5 4 5 5 5 5 4 5 5 5
[371] 5 5 5 5 5 5 5 4 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 4 5 5 5 5 5 5 4 4 4 5 5 4
[408] 4 4 4 4 3 4 4 4 4 4 3 4 4 4 4 4 4 4 4 4 4 4 4 5 4 3 3 3 4 4 4 3 3 3 3 3 3
[445] 3 3 4 4 5 5 5 5 5 5 4 5 5 5 5 5 5 5 4 5 5 5 5 5 5 5 5 5 4 5 5 5 5 5 5 5 5
[482] 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5
[519] 5 5 5 5 4 5 5 5 5 5 5 5 4 5 5 5 5 5 5 5 5 4 3 4 4 3 4 4 4 4 5 5 5 5 5 5 5
[556] 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 4 4 5 5 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 4
[593] 4 5 5 4 4 4 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 4 5 5 5 5 5 5 5 5 5 4 5
[630] 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 4 5 5 5 5 5 5 5 5 5 5
[667] 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 4 5 5 5 5 5 5 5 5 5 4 5 5 5 5 5 5 5 5 5 5 5
[704] 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
```

```
waiting_times
```

```
 [1]  0  0  0  0  0  0  0  0  1  2  2  7  8  5  0  0  0  0  0  0  0  0  0  0  0
[26]  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

```
 [51]  0  0  0  0  0  2  4  4  4  3  2  2  2  2  0  0  0  0  0  0  0  0  0  0  0
 [76]  0  0  0  0  0  0  2  4  4  6  3  1  1  0  0  0  1  0  0  0  2  1  2  3  5
[101]  3  3  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  1  0  0  0
[126]  0  0  0  0  0  0  0  0  0  0  2  1  5  3  2  0  3  2  4  5  2  2  4  4  4
[151]  4  4  3  3  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  1  1  6  6  6  6
[176]  6 10  8  8  9  7  9 10 16 21 21 26 26 24 27 29 28 28 29 31 31 30 30 36 35
[201] 35 32 32 32 32 32 27 26 27 27 28 35 38 38 40 40 39 39 40 40 40 41 42 45 46
[226] 46 47 44 42 41 41 40 37 33 30 23 26 28 30 30 28 29 29 31 31 30 27 27 27 26
[251] 27 28 29 26 26 29 32 32 32 32 34 34 34 34 31 32 32 32 31 33 33 33 32 32 33
[276] 39 39 42 43 46 48 49 47 48 47 50 50 49
```

Make plots using these three pieces of data? Tracked across each day How do the queueing times look?

Restaurant Profits

Assumptions:

1. each customer spends \$50 per meal (customers who are still in the queue when the restaurant closes won't pay)
2. each chef earns a wage of \$40 per hour (paid for the entire duration of the restaurant's operating hours)

Maximizing Profits

Should we run this simulation multiple times to create a PDF of the total daily profits? How many chefs should we hire?

Down-time of Restaurant

How does the occupancy of the restaurant vary throughout the day? Does that inform any of our recommendations?