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)
library(lubridate)
library(knitr)

set.seed(121)

# 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 value

arrival_times_s1 <- T_sample[T_sample <= total_time]

arrival_times_s1
```

```
 [1]  15.48044  19.99824  21.21837  37.74527  48.04027  56.58220  61.71317
 [8]  87.79562  90.61075  98.27931 104.41170 104.58245 106.38726 119.22966
[15] 129.63617 139.66808 156.91180 161.87999 195.55182 199.32194 199.90017
[22] 203.53702 207.00779 207.92471 210.76699 246.48358 255.32026 261.08139
[29] 267.57539 292.35047 334.71035 351.57806 355.43151 378.50418 389.27844
[36] 394.13197 394.80570 423.04531 426.98859 443.30627 445.14078 462.77508
[43] 469.07521 470.41177 482.74142 505.02956 511.81598 548.50360 548.97865
[50] 549.67378 550.72973 565.97825 588.13573 589.15853 591.93323 599.78691
[57] 603.96967 631.88036 653.39341 653.56135
```

```r
opening_time + minutes(floor(max(arrival_times_s1)))
```

```
[1] "10H 653M 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 15 minutes after opening and the last customer arriving 67 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_s1 <- rexp(length(arrival_times_s1), rate= lambdaS)

#these are the serving times for each arriving customer before they leave
service_times_s1
```

```
 [1] 18.3226535  3.6118119 23.3500006  6.1564912 72.6485441 11.8925547
 [7]  2.4963011 25.2319737  4.5310184 15.6495337  3.7841440  1.1254289
[13] 18.2791955 14.6349111 37.8510680  5.4804337  4.8154802  3.7401264
[19]  4.4097259  5.6360781  8.8267210  7.1504251  3.9439739 13.3167754
```

```
[25]  3.9331742  5.2054586  0.6097514  8.0874264 16.0986024  4.6551948
[31]  5.0889073  6.6017693 10.2738857 16.6394394  5.4733361 15.7197986
[37]  6.6819781  1.6205570 25.0606132  3.8022200  3.6576534 30.6646839
[43]  4.9999949  1.2557795  4.9764633  1.9951685 16.4762087  1.7785269
[49] 11.3563572  4.3010519  1.1843851 17.6175588  8.2691186  7.0229399
[55] 14.1578059 23.6575172  8.8494207 14.7084648  9.8187471  0.2995134
```

Time of the day with arrival time

```r
# Start time as POSIXct
start_time <- as.POSIXct("10:00", format = "%H:%M", tz = "UTC")

# Add minutes to the start time

time_of_day <- sapply(arrival_times_s1, function(m) {
  m <- round(m)  # Round to nearest whole number
  new_time <- start_time + (m * 60)  # Add minutes converted to seconds
  format(new_time, "%H:%M")  # Format as "HH:MM"
})

print(time_of_day)
```

```
 [1] "10:15" "10:20" "10:21" "10:38" "10:48" "10:57" "11:02" "11:28" "11:31"
[10] "11:38" "11:44" "11:45" "11:46" "11:59" "12:10" "12:20" "12:37" "12:42"
[19] "13:16" "13:19" "13:20" "13:24" "13:27" "13:28" "13:31" "14:06" "14:15"
[28] "14:21" "14:28" "14:52" "15:35" "15:52" "15:55" "16:19" "16:29" "16:34"
[37] "16:35" "17:03" "17:07" "17:23" "17:25" "17:43" "17:49" "17:50" "18:03"
[46] "18:25" "18:32" "19:09" "19:09" "19:10" "19:11" "19:26" "19:48" "19:49"
[55] "19:52" "20:00" "20:04" "20:32" "20:53" "20:54"
```

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 (a

# compare this to the arrival time
```

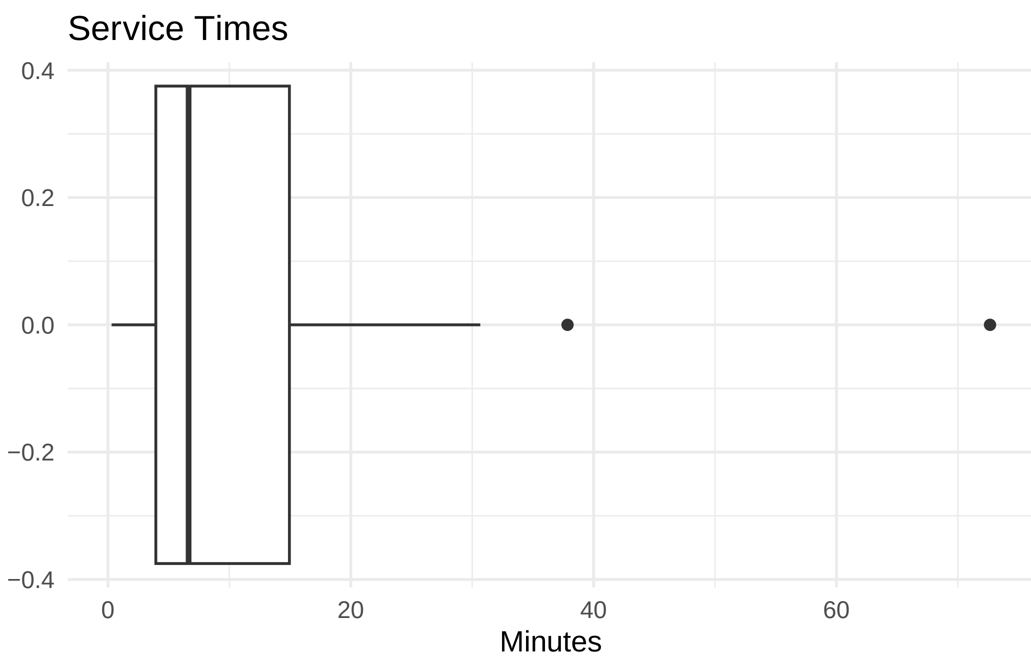
```r
# if arrival time > time service ends then wait time = 0
# but if arrival time < service time ends then wait time = time service ends- arrival time

# variable initialization
waiting_times_s1 <- numeric(length(arrival_times_s1))  # generating times for each customer
service_start <- numeric(length(arrival_times_s1))
service_end <- numeric(length(arrival_times_s1))
current_end <- numeric(0) # service end time for current customer (i)

# iterate over each customer
for (i in 1:length(arrival_times_s1)) {

  # 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_s1[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_s1[i]
  } else {
    # scenario 2: if there is a waiting time, service starts at the end of the previous custo
    previous_end <- service_end[i - 1]
    service_start[i] <- max(arrival_times_s1[i], previous_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_s1[i]

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

  # update waiting time
  waiting_times_s1[i] <- service_start[i] - arrival_times_s1[i]
}


scen1_sim_results <- data.frame(
  customer = 1:length(arrival_times_s1),
  arrival_time = arrival_times_s1,
  service_length = service_times_s1,
```

```
  service_start = service_start,
  service_end = service_end,
  waiting_time = waiting_times_s1,
  time_of_day = time_of_day
)

print(head(scen1_sim_results, 5)) # printing first 5 customers
```

```
  customer arrival_time service_length service_start service_end waiting_time
1        1     15.48044      18.322653      15.48044    33.80309      0.00000
2        2     19.99824       3.611812      33.80309    37.41490     13.80485
3        3     21.21837      23.350001      37.41490    60.76490     16.19654
4        4     37.74527       6.156491      60.76490    66.92140     23.01963
5        5     48.04027      72.648544      66.92140   139.56994     18.88113
  time_of_day
1       10:15
2       10:20
3       10:21
4       10:38
5       10:48
```

Serving and Waiting Times Analysis

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

scen1_sim_results %>%
  ggplot(aes(x= service_length)) +
  geom_boxplot() +
  labs(
    x= "Minutes",
    title = "Service Times"
  ) +
  theme_minimal()
```




```
mean(service_times_s1)
```

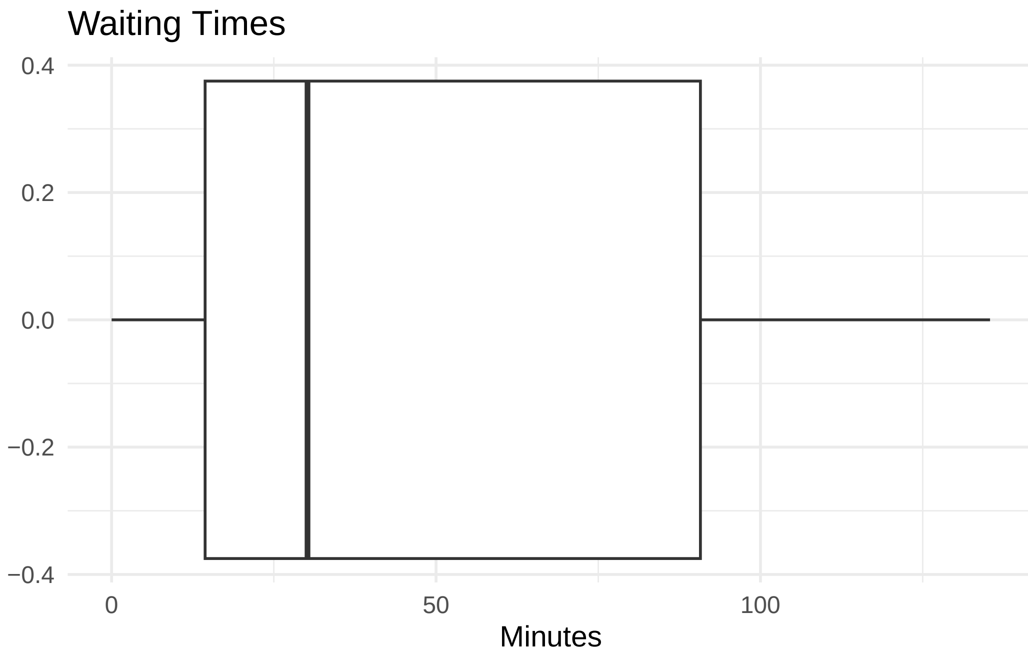
```
[1] 10.65808
```

The average service time is 11 minutes, with the data skewed right, consistent with an exponential distribution. This indicates that service times tend to lower.

```
# boxplot(waiting_times_s1, horizontal= TRUE, main= "Waiting Times", xlab= "Minutes")

mean(waiting_times_s1)
```

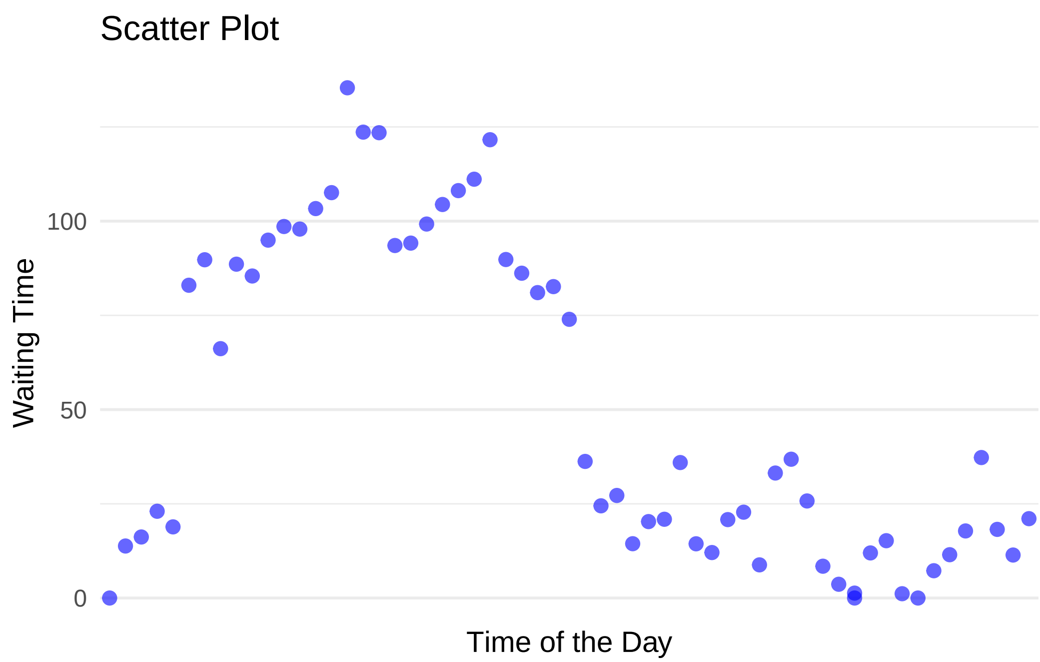
```
[1] 50.594
```

```
scen1_sim_results %>%
  ggplot(aes(x= waiting_time)) +
  geom_boxplot() +
  labs(
    x= "Minutes",
    title = "Waiting Times"
  ) +
  theme_minimal()
```

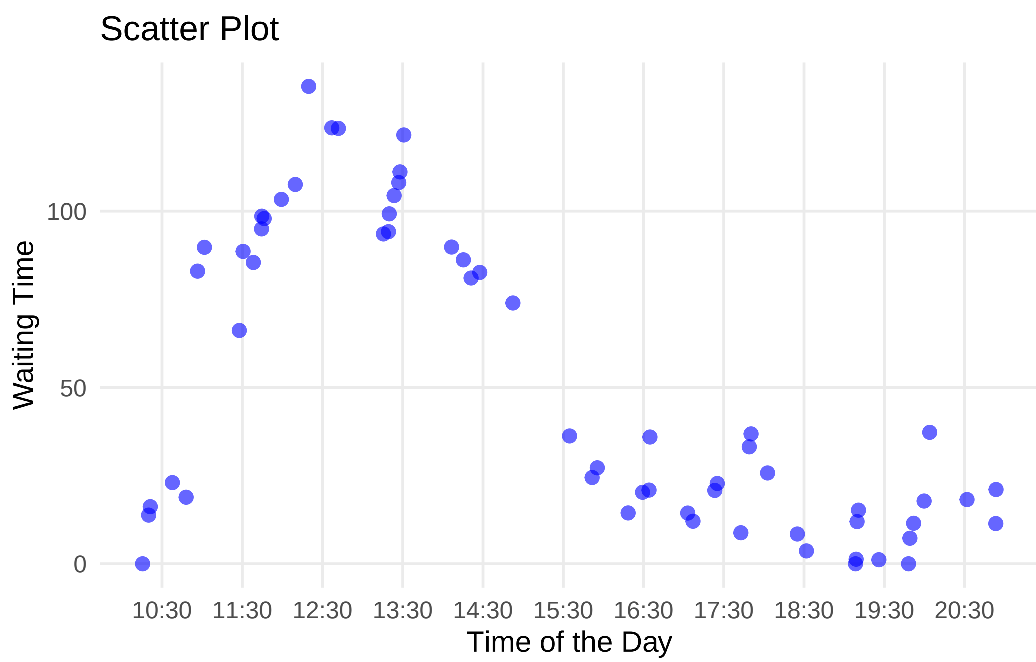


Waiting times tends to be slightly right-skewed and on average, the waiting time is 51 minutes.

```
#Label for 30 min interval
breaks <- seq(30, 720, by = 30)
labels <- sprintf("%02d:%02d", 10 + breaks %/% 60, breaks %% 60)

ggplot(scen1_sim_results, aes(x = time_of_day, y = waiting_times_s1)) +
  geom_point(color = "blue", size = 2, alpha = 0.6) +
    scale_x_discrete(
    breaks = breaks,
    labels = labels
  ) +
  labs(
    title = "Scatter Plot",
    x = "Time of the Day",
    y = "Waiting Time"
  ) +
  theme_minimal()
```




```
library(ggplot2)


# Custom breaks and labels for 30-minute intervals
breaks <- seq(30, 720, by = 60)
labels <- sprintf("%02d:%02d", 10 + breaks %/% 60, breaks %% 60)

# Scatter plot with x-axis as numeric time in minutes
ggplot(scen1_sim_results, aes(x = arrival_times_s1, y = waiting_times_s1)) +
  geom_point(color = "blue", size = 2, alpha = 0.6) +
  scale_x_continuous(
    breaks = breaks,
    labels = labels
  ) +
  labs(
    title = "Scatter Plot",
    x = "Time of the Day",
    y = "Waiting Time"
  ) +
  theme_minimal()+
  theme(panel.grid.minor = element_blank())
```



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)**

```
# 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
num_chefs = 2

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 = num_chefs)
```

Waiting Times

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

```
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)

# 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)
```

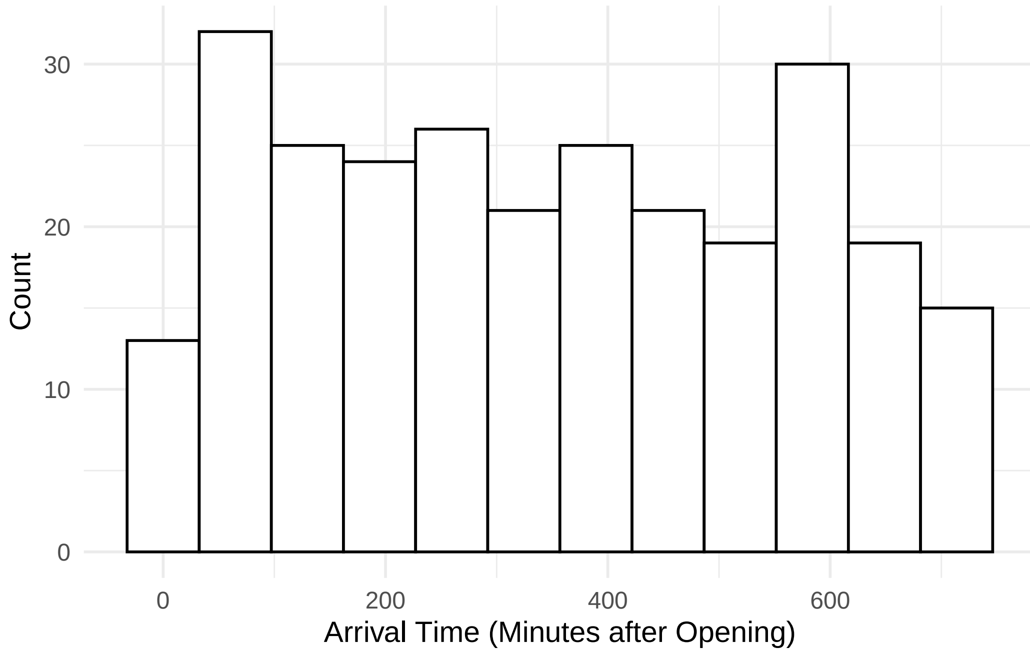
```
# the arrival_times indices of the people currently in line
# in order to know how long their eventual service time will be
queue <- numeric(0)

# 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
    # 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
```

```r
    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)
}

occupied_tables_history <- occupied_tables_history[-1]
```

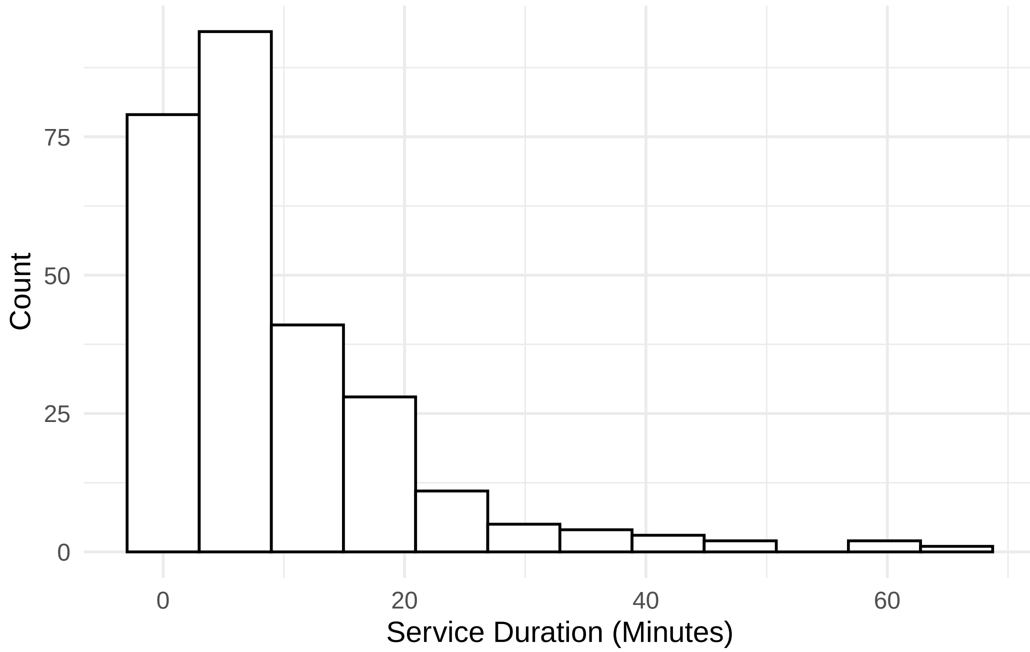
```r
scen2_sim_results_by_customer <- data.frame(
  customer = 1:length(arrival_times),
  arrival_time = arrival_times,
  service_length = service_times,
  waiting_time = waiting_times
)
```

```r
scen2_sim_results_by_minute <- data.frame(
  minutes_since_opening = 1:total_time,
  time_of_day = I(lapply(1:total_time, function(i) opening_time + minutes(i))),
  queue_size = queue_size_history,
  occupied_tables = occupied_tables_history
)
```

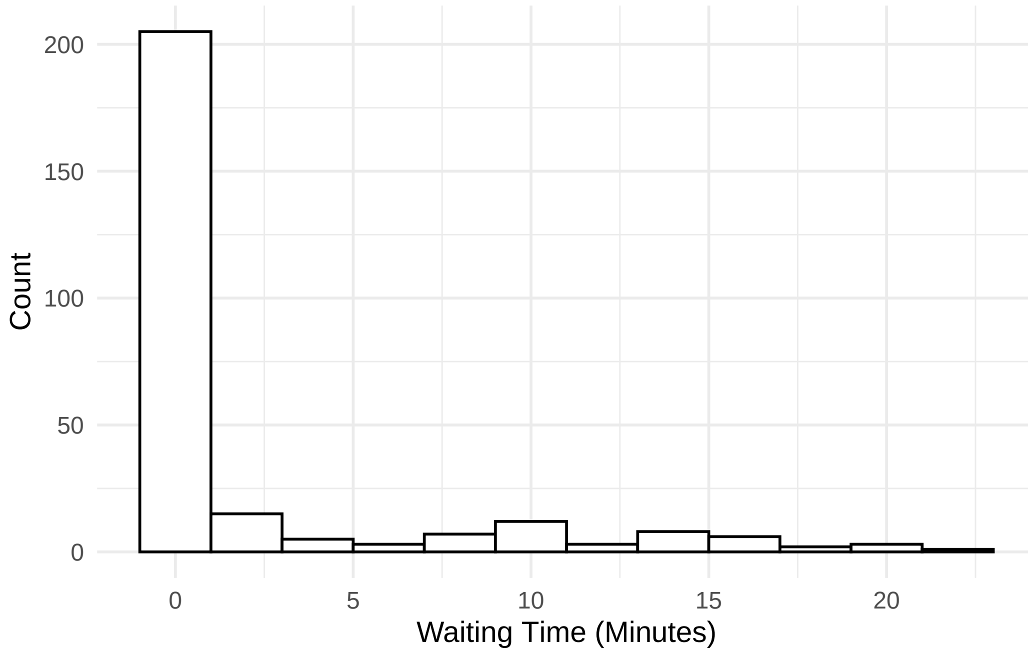
```r
scen2_sim_results_by_customer |>
  ggplot(aes(x = arrival_time)) +
  geom_histogram(bins = 12, color = "black", fill = "white") +
  labs(
    x = "Arrival Time (Minutes after Opening)",
    y = "Count"
  ) +
  theme_minimal()
```



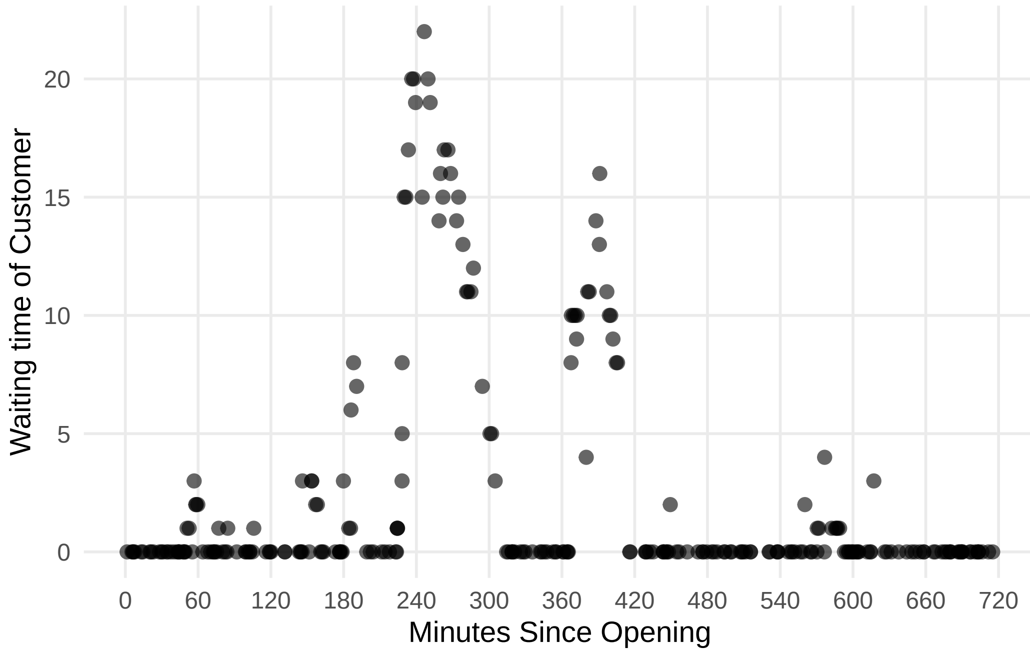

```r
scen2_sim_results_by_customer |>
  ggplot(aes(x = service_length)) +
  geom_histogram(bins = 12, color = "black", fill = "white") +
  labs(
    x = "Service Duration (Minutes)",
    y = "Count"
  ) +
  theme_minimal()
```



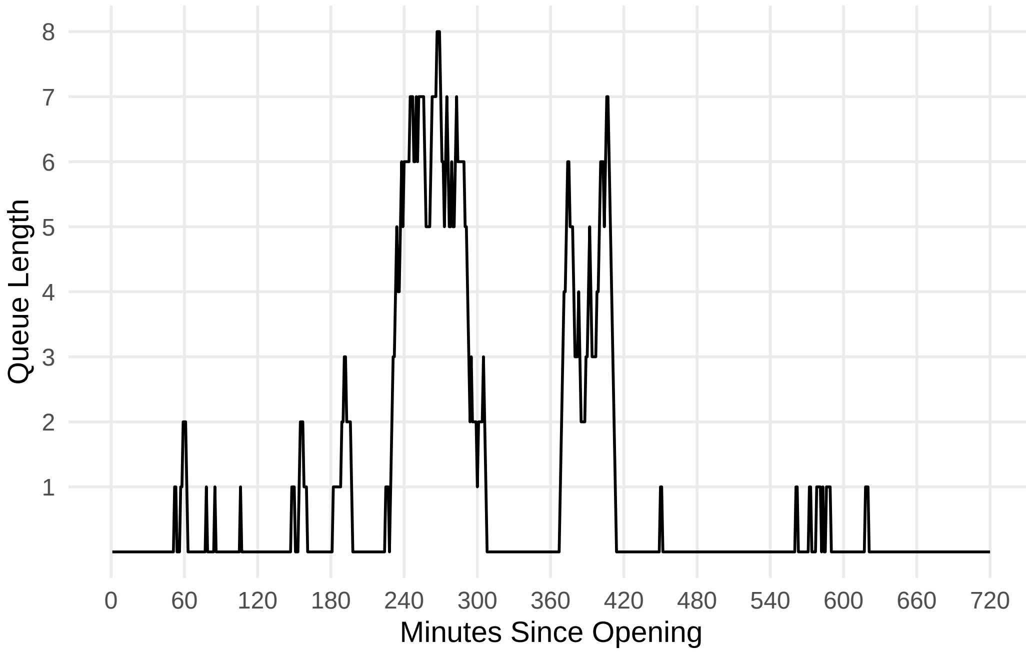

```
scen2_sim_results_by_customer |>
  ggplot(aes(x = waiting_time)) +
  geom_histogram(bins = 12, color = "black", fill = "white") +
  labs(
    x = "Waiting Time (Minutes)",
    y = "Count"
  ) +
  theme_minimal()
```



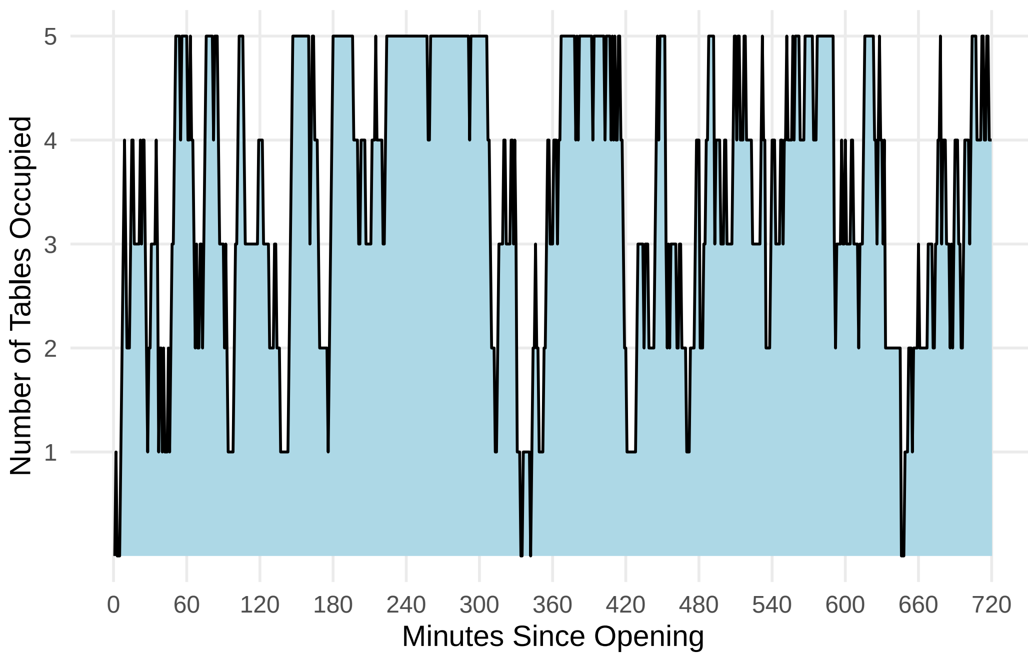

```
scen2_sim_results_by_customer |>
  ggplot(aes(x = arrival_time, y = waiting_time)) +
  geom_point(size = 2, alpha = 0.6) +
  scale_x_continuous(breaks = seq(0, total_time, by = 60)) +
  labs(
    x = "Minutes Since Opening",
    y = "Waiting time of Customer"
  ) +
  theme_minimal() +
  theme(panel.grid.minor = element_blank())
```




```
scen2_sim_results_by_minute |>
  ggplot(aes(x = minutes_since_opening, y = queue_size)) +
  geom_line() +
  scale_y_continuous(breaks = seq(1, max(queue_size_history), by = 1)) +
  scale_x_continuous(breaks = seq(0, total_time, by = 60)) +
  labs(
    x = "Minutes Since Opening",
    y = "Queue Length"
  ) +
  theme_minimal() +
  theme(panel.grid.minor = element_blank())
```




```
scen2_sim_results_by_minute |>
  ggplot(aes(x = minutes_since_opening, y = occupied_tables)) +
  geom_area(fill = "lightblue") +
  geom_line() +
  scale_y_continuous(breaks = seq(1, tables, by = 1)) +
  scale_x_continuous(breaks = seq(0, total_time, by = 60)) +
  labs(
    x = "Minutes Since Opening",
    y = "Number of Tables Occupied"
  ) +
  theme_minimal() +
  theme(panel.grid.minor = element_blank())
```



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)
3. Customer will not wait longer than 30 minutes

```
# assumption of five tables and 3 chefs

# setting number of chefs to a certain number
# chefs <- 3
# meal <- 50 #50 dollars per meal
# wage <- 12*40 #40 dollars per hour for 12 hrs of work
# revenue <- total_customers * meal
# costs <- wage*chefs
# profit <- revenue - costs
# profit
```

```r
# only including customers who are waiting for less than 30 mins because will leave the line

filtered_customers <- which(waiting_times < 30)
total_customers <- length(filtered_customers)
total_customers
```

```
[1] 270
```

```r
meal <- 50 #50 dollars per meal
wage <- 12*40 #40 dollars per hour for 12 hrs of work

profit_table <- data.frame(
  Chefs = integer(),
  Revenue = numeric(),
  Costs = numeric(),
  Profit = numeric()
)

# if people get mad, the meal will only be 25 because they are mad

for (chefs in 1:10) {

  revenue <- total_customers * meal

  # Calculate costs (wage per chef * number of chefs)
  costs <- wage * chefs

  # Calculate profit
  profit <- revenue - costs

  profit_table <- rbind(profit_table, data.frame(Chefs = chefs, Revenue = revenue, Costs = co

}

print(profit_table)
```

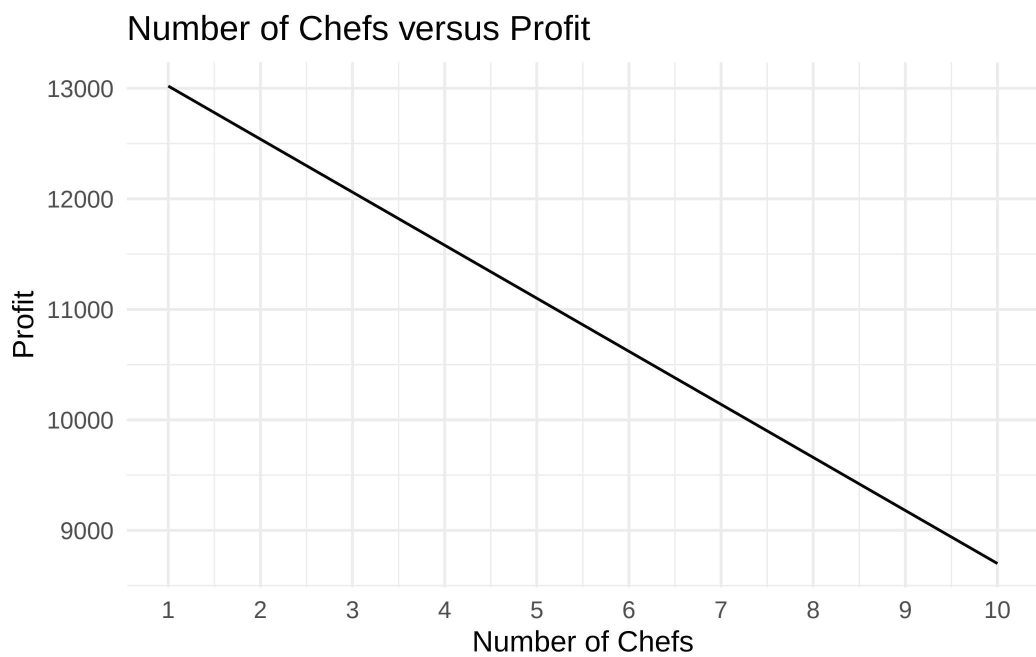
```
   Chefs Revenue Costs Profit
1      1   13500   480  13020
2      2   13500   960  12540
3      3   13500  1440  12060
4      4   13500  1920  11580
```

```
5       5    13500   2400   11100
6       6    13500   2880   10620
7       7    13500   3360   10140
8       8    13500   3840    9660
9       9    13500   4320    9180
10     10    13500   4800    8700
```

```
profit_table
```

```
   Chefs Revenue Costs Profit
1       1    13500    480   13020
2       2    13500    960   12540
3       3    13500   1440   12060
4       4    13500   1920   11580
5       5    13500   2400   11100
6       6    13500   2880   10620
7       7    13500   3360   10140
8       8    13500   3840    9660
9       9    13500   4320    9180
10     10    13500   4800    8700
```

```
profit_table %>%
  ggplot(aes(x = Chefs, y = Profit)) +
  geom_line() +
  scale_x_continuous(breaks = seq(1, 10, 1))+
  labs(
    x= "Number of Chefs",
    y= "Profit",
    title= "Number of Chefs versus Profit"
  ) +
  theme_minimal()
```



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

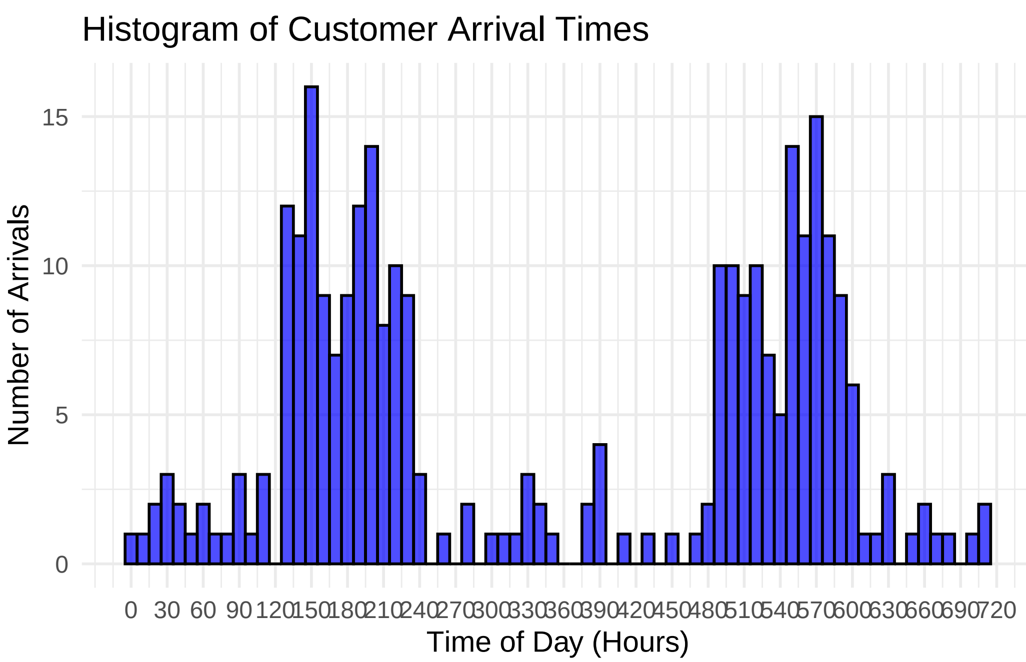
How does the occupancy of the restaurant vary throughout the day? Does that inform any of our recommendations?

Scenario 3

```
simulate_differential_arrival_times <- function(
    lunch_peak_start,lunch_peak_end,
    dinner_peak_start, dinner_peak_end,
    lambda_down, lambda_peak, total_time
) {
  # Convert arrival rates to per minute
  rate_down <- lambda_down / 60
```

```
  rate_peak <- lambda_peak / 60

  # Convert peak times to minutes from opening
  lunch_peak_start_min <- as.numeric(as.duration(lunch_peak_start - opening_time), units = "m
  lunch_peak_end_min <- as.numeric(as.duration(lunch_peak_end - opening_time), units = "minut

  dinner_peak_start_min <- as.numeric(as.duration(dinner_peak_start - opening_time), units =
  dinner_peak_end_min <- as.numeric(as.duration(dinner_peak_end - opening_time), units = "mir

  # Initialize list to store arrival times
  arrival_times <- numeric()
  current_time <- 0  # Start at 0 minutes (opening time)

  # Generate arrival times
  while (current_time < total_time) {
    # Determine the arrival rate based on current time
    if ((current_time >= lunch_peak_start_min && current_time < lunch_peak_end_min) ||
        (current_time >= dinner_peak_start_min && current_time < dinner_peak_end_min)) {
      arrival_rate <- rate_peak  # Peak time rate
    } else {
      arrival_rate <- rate_down  # Downtime rate
    }

    # Generate the next interarrival time from the exponential distribution
    next_arrival <- rexp(1, arrival_rate)

    # Update the current time
    current_time <- current_time + next_arrival

    # If within the operating hours, add the arrival time to the list
    if (current_time < total_time) {
      arrival_times <- c(arrival_times, current_time)
    }
  }

  # Return arrival_times
  return(arrival_times)
}

arrival_times <- simulate_differential_arrival_times(
  lunch_peak_start = hm("12:00"),
  lunch_peak_end = hm("14:00"),
```

```
  dinner_peak_start = hm("18:00"),
  dinner_peak_end = hm("20:00"),
  lambda_down = 6, lambda_peak = 60, total_time = total_time)

ggplot(data = data.frame(arrival_times), aes(x = arrival_times)) +
  geom_histogram(binwidth = 10, fill = "blue", color = "black", alpha = 0.7) +
  labs(
    title = "Histogram of Customer Arrival Times",
    x = "Time of Day (Hours)",
    y = "Number of Arrivals"
  ) +
  scale_x_continuous(
    breaks = seq(0, 720, by = 30)
  ) +
  theme_minimal()
```




```
restaurant_sim <- function(arrivals, chefs, tables, minutes) {
  # calculate service times
  service_times = rexp(length(arrivals), rate = (3*chefs) / 60)

  # set up tracking
  arrival_times_temp <- arrival_times
```

```
queue_size_history <- numeric(total_time)

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

# 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

# iterate through the day
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)
```

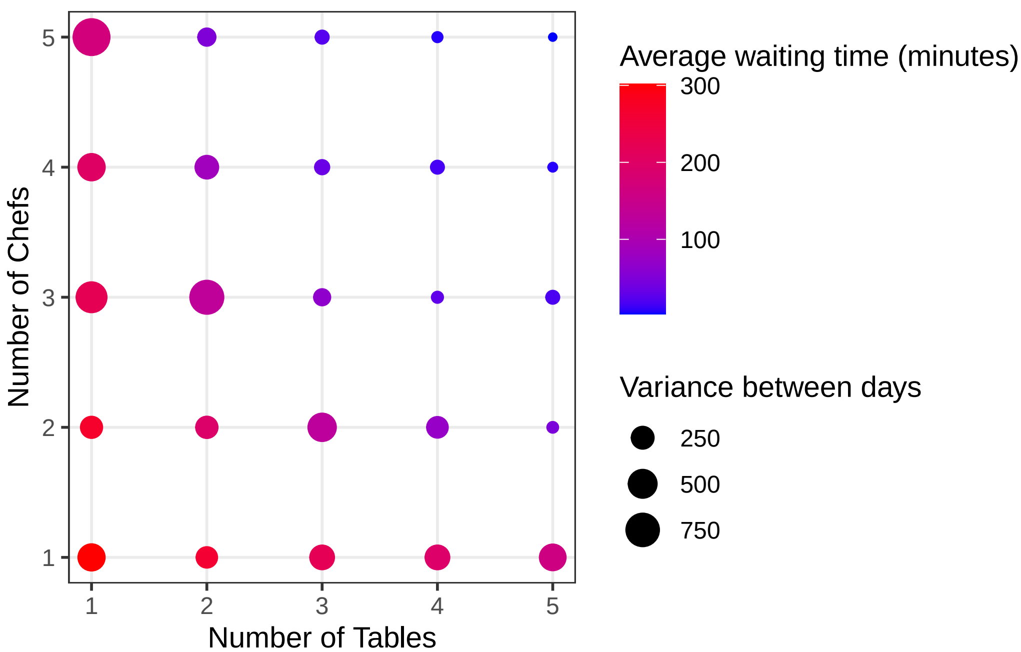
```r
      customers_entered = customers_entered + 1
    }
  }
  # are any tables currently open and there is a person in line?
  if(occupied_tables_history[i+1] < tables & length(queue) > 0) {
    # if so, then seat the first person in line
    # at the first available table
    for (j in 1:tables)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

occupied_tables_history <- occupied_tables_history[-1]

# calculate outputs (the things we actually care about) from the simulation

# average waiting time across all customers
avg_waiting_time <- mean(waiting_times)
# number of customers who waited >30 minutes (and made us less money)
long_waits <- length(waiting_times[waiting_times > 30])
# average queue length throughout the day
avg_queue_length <- mean(queue_size_history)
# maximum queue length that day
max_queue_length <- max(queue_size_history)
# average table occupancy in the restaurant
avg_tables_occupied <- mean(occupied_tables_history)
# number of customers
```

```r
  num_customers <- length(waiting_times)

  # return all of it, as a data frame with one row
  sim_output <- data.frame(
    total_customers = num_customers,
    num_chefs = chefs,
    num_tables = tables,
    avg_waiting_time = avg_waiting_time,
    long_waits = long_waits,
    avg_queue_length = avg_queue_length,
    max_queue_length = max_queue_length,
    avg_tables_occupied = avg_tables_occupied
  )
  return(sim_output)
}

total_time <- 720

df <- numeric(8)
for(a in 1:5) {
  num_chefs = a
  for(b in 1:5) {
    num_tables = b
    for(i in 1:10) {
      arrival_times <- simulate_differential_arrival_times(
      lunch_peak_start = hm("12:00"),
      lunch_peak_end = hm("14:00"),
      dinner_peak_start = hm("18:00"),
      dinner_peak_end = hm("20:00"),
      lambda_down = 6, lambda_peak = 60, total_time = total_time)
      df <- rbind(df, restaurant_sim(arrival_times, num_chefs, num_tables, total_time))
    }
  }
}

df <- df[-1, ]
```

```r
# MEAN WAITING TIMES
df |>
  group_by(num_chefs, num_tables) |>
  summarise(mean = mean(avg_waiting_time), variance = var(avg_waiting_time)) |>
  ggplot(aes(x = num_tables, y = num_chefs, color = mean, size = variance)) +
```

```
  geom_point() +
  scale_color_gradient(low = "blue", high = "red") +
  scale_x_continuous(
    breaks = seq(min(df$num_tables), ceiling(max(df$num_tables)), by = 1),
    labels = seq(min(df$num_tables), ceiling(max(df$num_tables)), by = 1)
  ) +
  scale_y_continuous(
    breaks = seq(min(df$num_chefs), ceiling(max(df$num_chefs)), by = 1),
    labels = seq(min(df$num_chefs), ceiling(max(df$num_chefs)), by = 1)
  ) +
  theme_bw() +
  theme(panel.grid.minor = element_blank()) +
  labs(
    x = "Number of Tables",
    y = "Number of Chefs",
    color = "Average waiting time (minutes)",
    size = "Variance between days"
  )
```

```
`summarise()` has grouped output by 'num_chefs'. You can override using the
`.groups` argument.
```



```r
# MEAN LONGEST WAITING TIME OF THE DAY
df |>
  group_by(num_chefs, num_tables) |>
  summarise(mean = mean(long_waits), variance = var(long_waits)) |>
  kable()
```

```
`summarise()` has grouped output by 'num_chefs'. You can override using the
`.groups` argument.
```

num_chefs	num_tables	mean	variance
1	1	263.9	377.21111
1	2	261.9	400.54444
1	3	242.6	315.37778
1	4	247.5	471.61111
1	5	235.6	385.37778
2	1	252.3	583.12222
2	2	254.0	73.33333
2	3	235.9	837.65556
2	4	197.0	704.44444
2	5	167.4	366.71111
3	1	248.1	331.43333
3	2	234.0	590.22222
3	3	203.5	1612.50000
3	4	124.0	453.55556
3	5	71.3	1606.45556
4	1	249.6	740.93333
4	2	229.5	872.72222
4	3	137.9	1052.98889
4	4	47.5	2012.50000
4	5	4.9	153.43333
5	1	243.3	856.45556
5	2	172.9	1085.21111
5	3	93.4	1661.37778
5	4	2.1	35.65556
5	5	0.0	0.00000

```r
# MEAN QUEUE LENGTH
df |>
  group_by(num_chefs, num_tables) |>
```

```r
  summarise(mean = mean(avg_queue_length), variance = var(avg_queue_length)) |>
  kable()
```

```
`summarise()` has grouped output by 'num_chefs'. You can override using the
`.groups` argument.
```

num_chefs	num_tables	mean	variance
1	1	114.430972	201.5884962
1	2	101.103056	76.8142497
1	3	80.611250	87.8268898
1	4	73.945833	70.4323508
1	5	59.906667	68.5130443
2	1	99.815139	135.5648090
2	2	74.871528	45.6540442
2	3	48.793333	144.6835146
2	4	26.689861	33.3788392
2	5	18.119444	2.3787697
3	1	83.855972	159.6868116
3	2	48.393194	146.5815400
3	3	26.372361	33.7116729
3	4	11.038750	4.4214045
3	5	7.094444	5.8615629
4	1	76.572917	145.5822129
4	2	34.004583	73.5809238
4	3	12.979167	10.0737444
4	4	6.169861	7.3625052
4	5	2.781528	0.5663653
5	1	64.822222	266.2274704
5	2	19.521389	21.9349139
5	3	8.757222	6.4968522
5	4	2.769861	1.7760846
5	5	1.302222	0.2660122

```r
# MEAN MAX QUEUE LENGTH FOR EACH DAY
df |>
  group_by(num_chefs, num_tables) |>
  summarise(mean = mean(max_queue_length), variance = var(max_queue_length)) |>
  kable()
```

```
`summarise()` has grouped output by 'num_chefs'. You can override using the
```

```
`.groups` argument.
```

num_chefs	num_tables	mean	variance
1	1	238.0	283.11111
1	2	207.4	368.04444
1	3	166.8	486.84444
1	4	156.0	383.33333
1	5	123.9	239.87778
2	1	202.3	491.34444
2	2	157.4	173.37778
2	3	110.9	619.43333
2	4	72.6	143.82222
2	5	59.4	54.26667
3	1	175.8	560.84444
3	2	104.3	336.67778
3	3	68.9	67.21111
3	4	44.4	56.04444
3	5	37.2	84.84444
4	1	154.8	629.95556
4	2	80.4	304.93333
4	3	48.9	48.98889
4	4	31.5	101.16667
4	5	21.0	38.44444
5	1	133.7	953.56667
5	2	59.7	80.67778
5	3	39.0	51.77778
5	4	19.2	33.51111
5	5	9.9	11.21111

```r
# MEAN TABLES OCCUPIED
df |>
  group_by(num_chefs, num_tables) |>
  summarise(mean = mean(avg_tables_occupied), variance = var(avg_tables_occupied)) |>
  kable()
```

```
`summarise()` has grouped output by 'num_chefs'. You can override using the
`.groups` argument.
```

num_chefs	num_tables	mean	variance
1	1	0.9645833	0.0007408
1	2	1.8750000	0.0027178
1	3	2.7327778	0.0115896
1	4	3.5940278	0.0159435
1	5	4.3398611	0.0074254
2	1	0.9484722	0.0019271
2	2	1.7775000	0.0022607
2	3	2.5745833	0.0062955
2	4	3.1355556	0.0717186
2	5	3.4790278	0.0525418
3	1	0.9254167	0.0013212
3	2	1.7100000	0.0031189
3	3	2.3684722	0.0143086
3	4	2.4168056	0.0128834
3	5	2.5072222	0.0593415
4	1	0.8977778	0.0011043
4	2	1.6798611	0.0040442
4	3	1.8815278	0.0166384
4	4	1.9202778	0.0369137
4	5	1.8926389	0.0046974
5	1	0.9030556	0.0011899
5	2	1.4329167	0.0115908
5	3	1.6111111	0.0137509
5	4	1.4969444	0.0170545
5	5	1.6027778	0.0307999