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

Open at 10am, close at 10pm, 5 customers arrive per hour on average

(Expressed in minutes after opening)

```
 [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
```

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

The average customer takes 1/6 hours, or 10 minutes to serve. So $= 6$

(The number of minutes taken by each customer after sitting down in the restaurant)

```
[1] 6
```

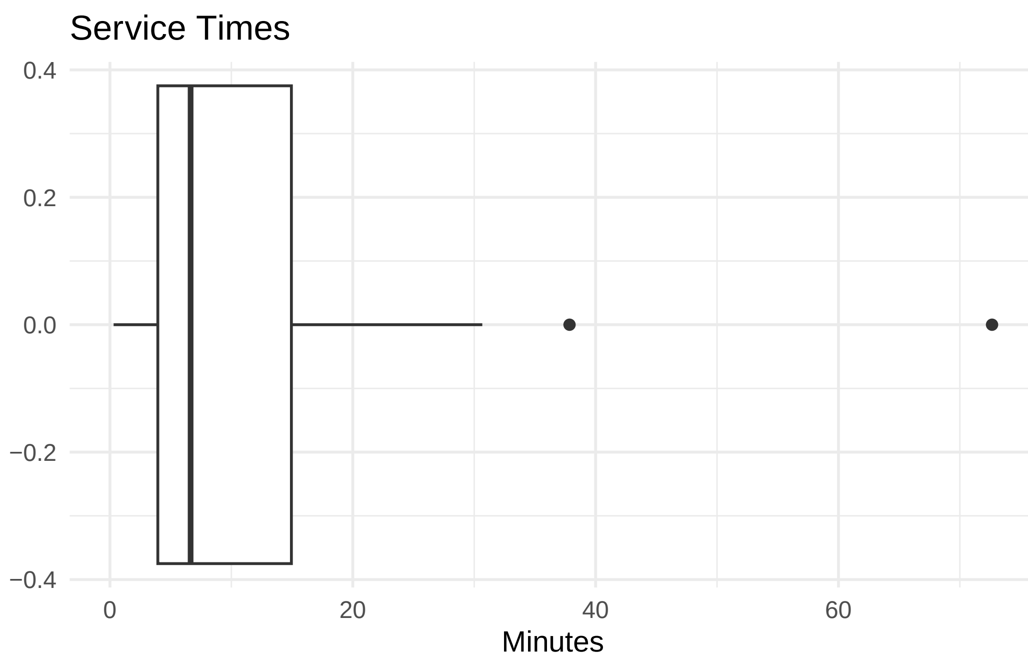
Time of the day with arrival time

```
 [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

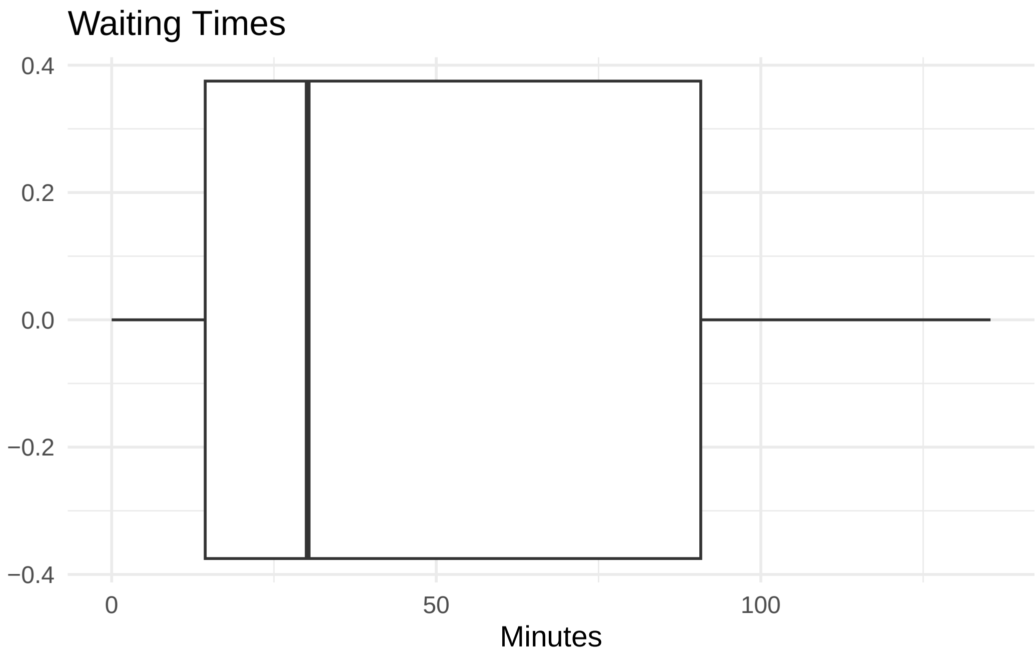
```
  customer arrival_time service_length service_start service_end waiting_time
1        1     15.48044      18.322653      15.48044    33.80309      0.00000
2        2     19.99824       3.611812      33.80309    37.41490     13.80485
3        3     21.21837      23.350001      37.41490    60.76490     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



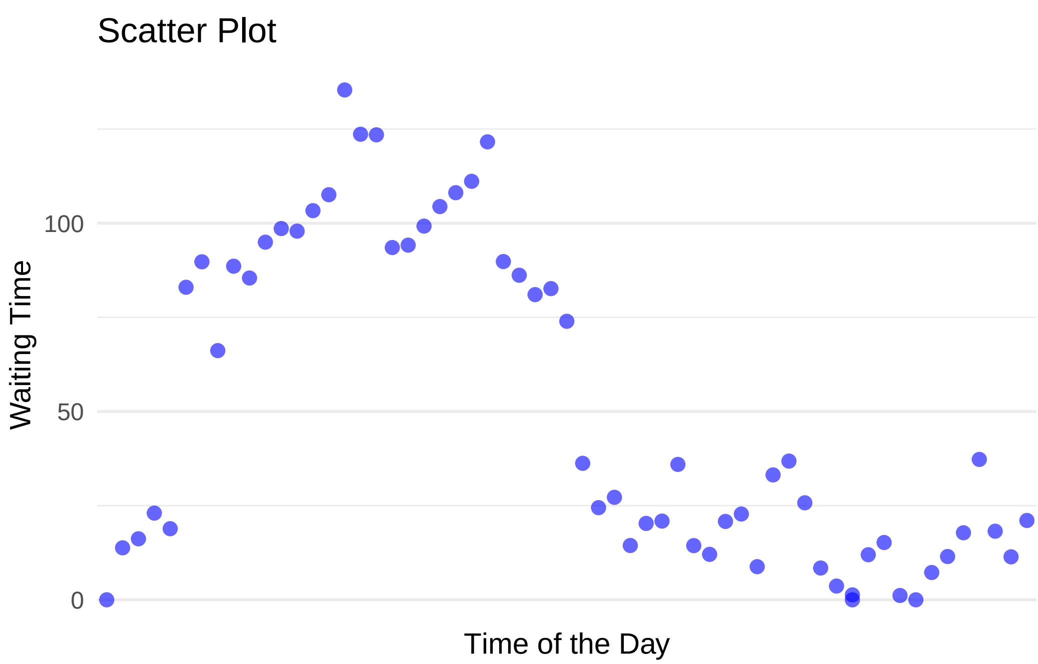

```
[1] 10.65808
```

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.




```
[1] 50.594
```

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



Scenario 2

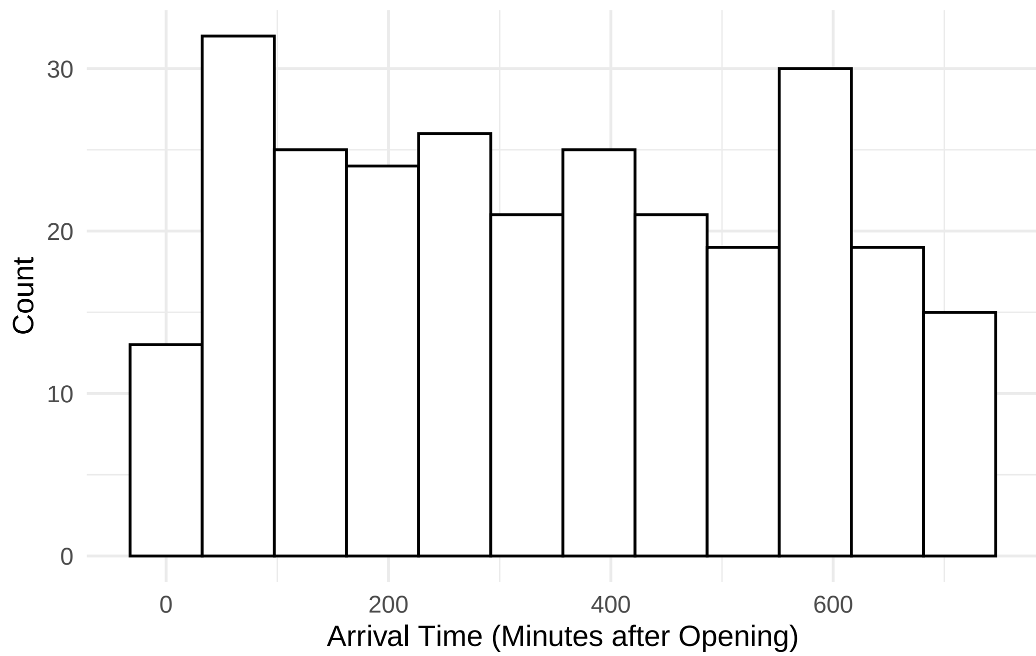
Assumptions:

1. 5 dining tables and L chefs with operating hours 10am - 10pm. We choose here that L = 2
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
4. 24 customers arrive every hour

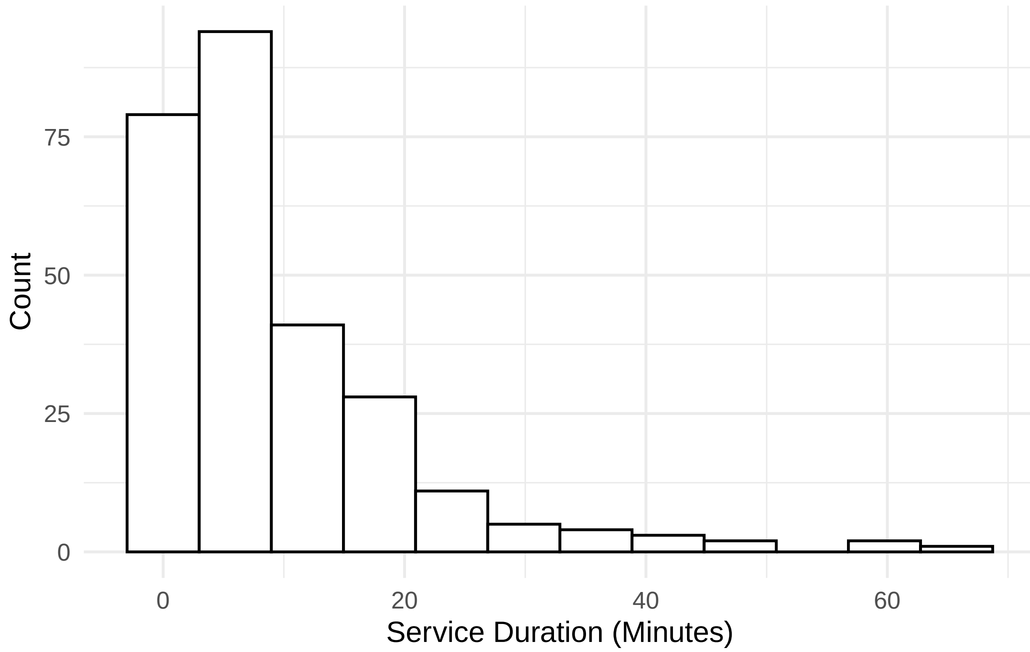
Waiting Times

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

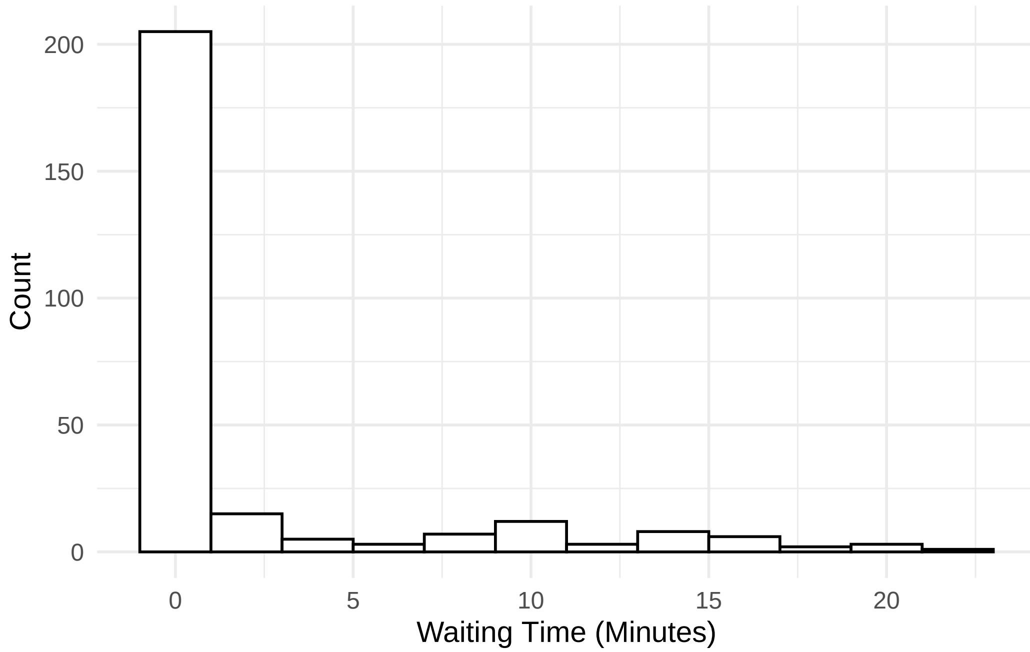
```
scen2_sim_results_by_customer |>
  ggplot(aes(x = arrival_time)) +
  geom_histogram(bins = 12, color = "black", fill = "white") +
  labs(
    x = "Arrival Time (Minutes after Opening)",
    y = "Count"
  ) +
  theme_minimal()
```



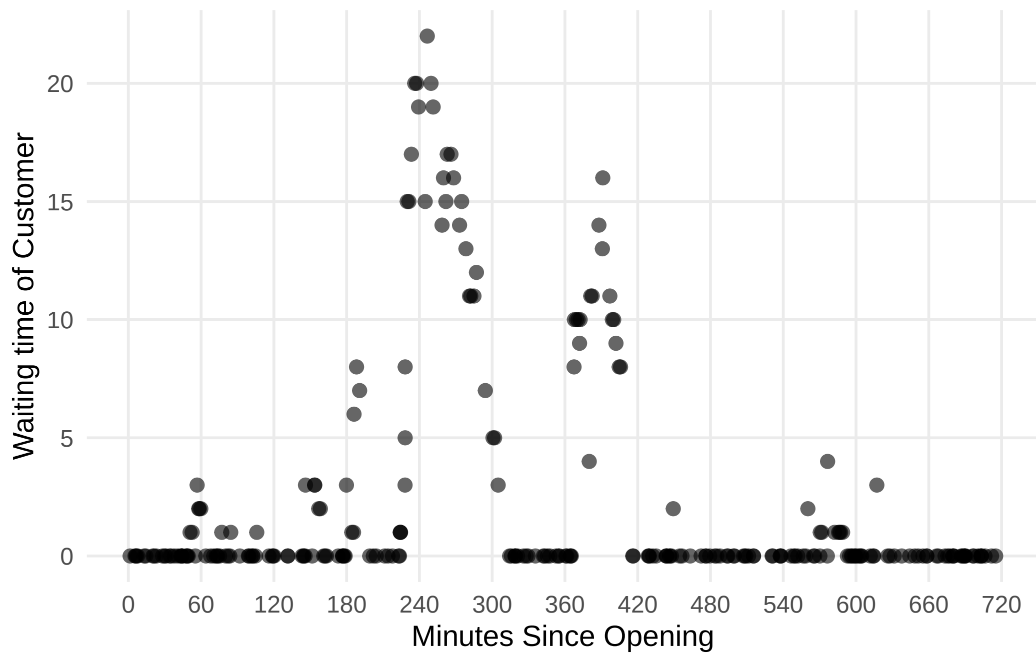

```r
scen2_sim_results_by_customer |>
  ggplot(aes(x = service_length)) +
  geom_histogram(bins = 12, color = "black", fill = "white") +
  labs(
    x = "Service Duration (Minutes)",
    y = "Count"
  ) +
  theme_minimal()
```



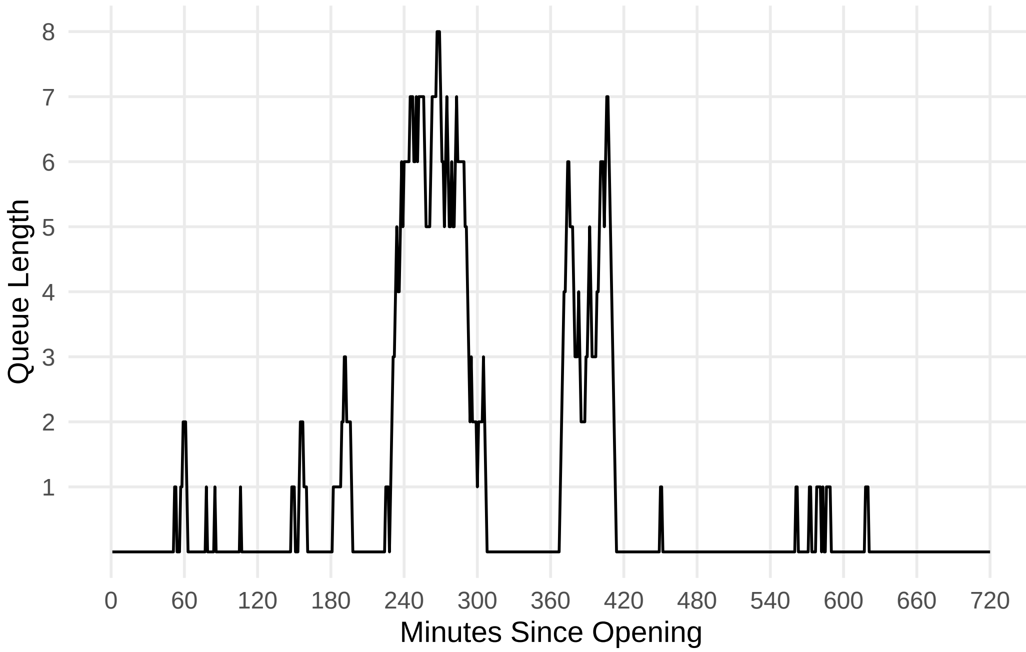

```
scen2_sim_results_by_customer |>
  ggplot(aes(x = waiting_time)) +
  geom_histogram(bins = 12, color = "black", fill = "white") +
  labs(
    x = "Waiting Time (Minutes)",
    y = "Count"
  ) +
  theme_minimal()
```



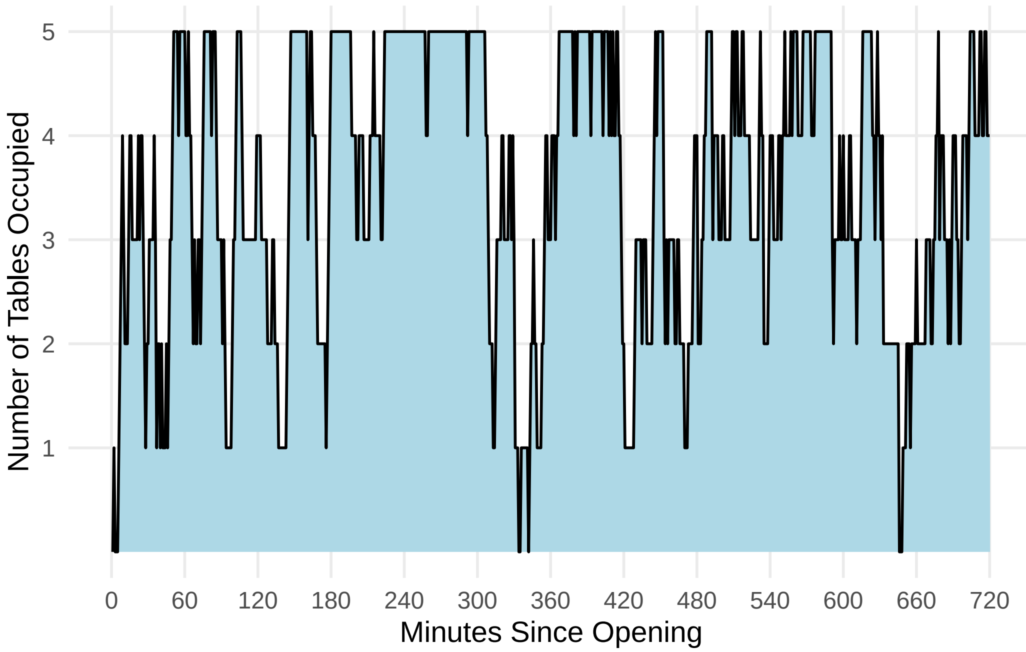

```r
scen2_sim_results_by_customer |>
  ggplot(aes(x = arrival_time, y = waiting_time)) +
  geom_point(size = 2, alpha = 0.6) +
  scale_x_continuous(breaks = seq(0, as.numeric(total_time_s2), by = 60)) +
  labs(
    x = "Minutes Since Opening",
    y = "Waiting time of Customer"
  ) +
  theme_minimal() +
  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(sim_s2$queue_size_history), by = 1)) +
  scale_x_continuous(breaks = seq(0, as.numeric(total_time_s2), by = 60)) +
  labs(
    x = "Minutes Since Opening",
    y = "Queue Length"
  ) +
  theme_minimal() +
  theme(panel.grid.minor = element_blank())
```




```r
# OCCUPIED TABLES
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, num_tables_s2, by = 1)) +
  scale_x_continuous(breaks = seq(0, as.numeric(total_time_s2), by = 60)) +
  labs(
    x = "Minutes Since Opening",
    y = "Number of Tables Occupied"
  ) +
  theme_minimal() +
  theme(panel.grid.minor = element_blank())
```



Restaurant Profits

Assumptions:

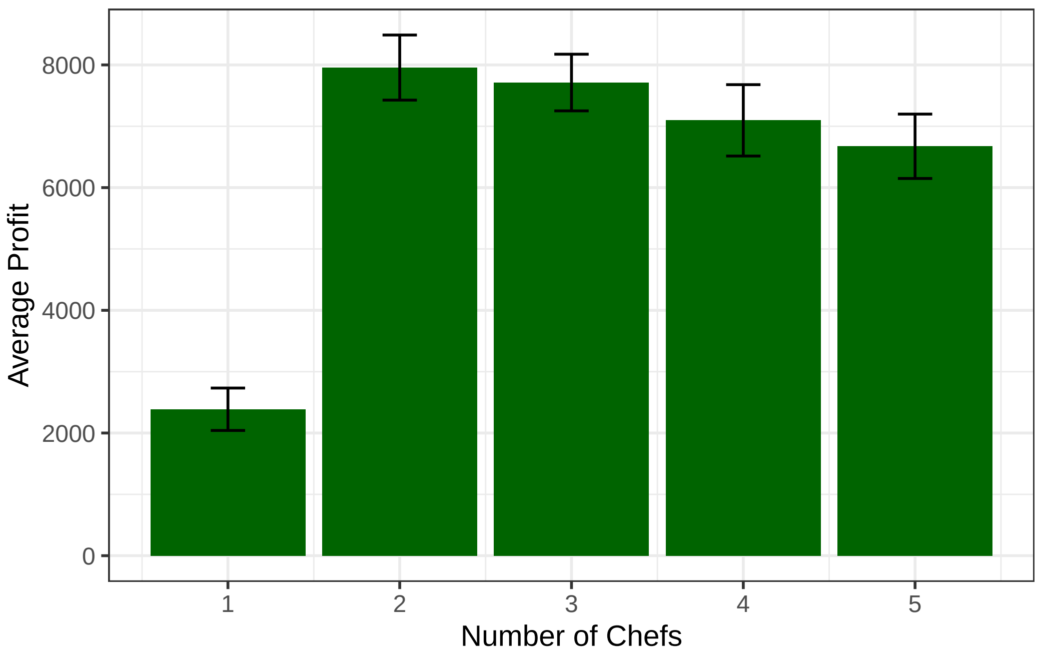
1. each customer spends $50 per meal (customers who are still in the queue when the restaurant closes won't pay)
2. each chef earns a wage of $40 per hour (paid for the entire duration of the restaurant's operating hours)
3. Each table cost $1000 per day (extra service cost, rent, etc.)
4. For customers who waited more than 30 minutes, they earn the restaurant half the amount of customers who didn't.

Maximizing Profits

With 5 tables, 24 customers arriving per hour, and these dollar amounts, how many chefs should we hire? We will run our simulation 100 times with 1 to 5 chefs on staff, to see which will maximize the expected profit.

	total_customers	profit	num_chefs	num_tables	avg_waiting_time	long_waits
1	288	1995	1	5	187.55208333	277
2	282	7180	4	5	0.10638298	0
3	282	6700	5	5	0.00000000	0
4	264	7240	2	5	3.57196970	0
5	288	7480	4	5	0.05555556	0
6	288	2320	1	5	164.96527778	264
7	284	7280	4	5	0.04577465	0
8	288	7960	3	5	0.56250000	0

	avg_queue_length	max_queue_length	avg_tables_occupied
1	75.02083333	129	4.966667
2	0.04166667	3	2.023611
3	0.00000000	0	1.495833
4	1.30972222	12	3.588889
5	0.02222222	2	1.806944
6	65.98611111	117	4.925000
7	0.01805556	2	2.134722
8	0.22500000	4	2.783333

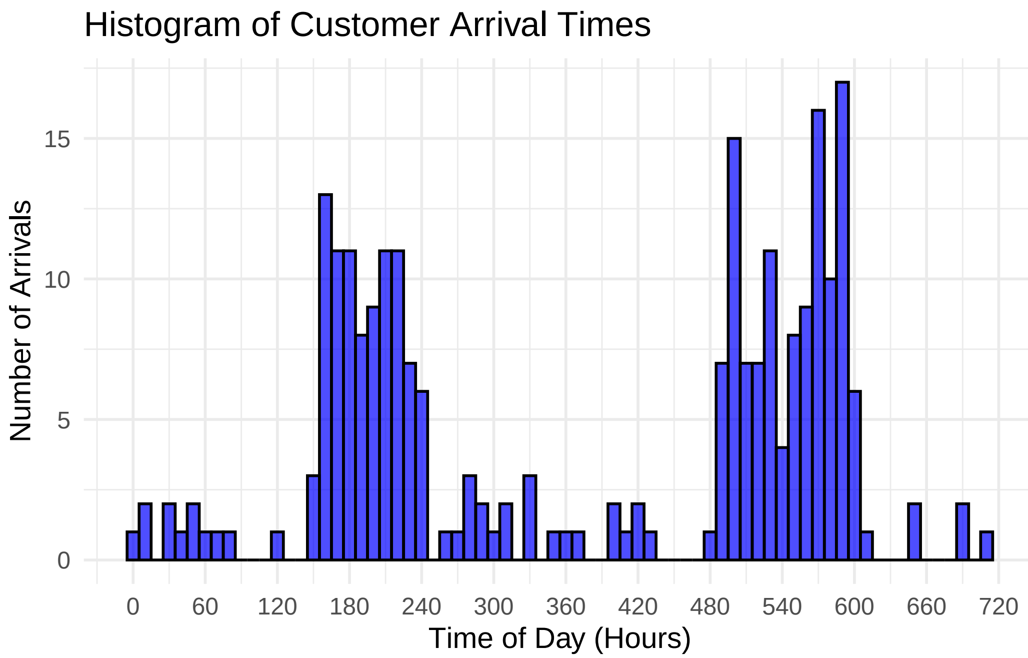


Scenario 3

To make the simulation more realistic, we have a third scenario.

Assumptions: 1. Open at 10am, close at 10pm 2. From 12pm to 2pm and 6pm to 8pm, 60 customers arrive every hour. Otherwise, 6 arrive every hour. 3. Instead of simulating service times with Exp( ) where = 3 times the number of chefs, we do = ln(chefs + 1), so that additional chefs beyond 2 make more of an impact. 4. Each customer will sit for a minimum of 45 minutes. This flat value will be added to the simulated service time, and is unaffected by staffing. 5. In the profit calculation, there is a cost of adding additional tables (which are now variable), which is $40 per table. 6. Chefs still cost $40 per hour to hire, and each customer earns $50.

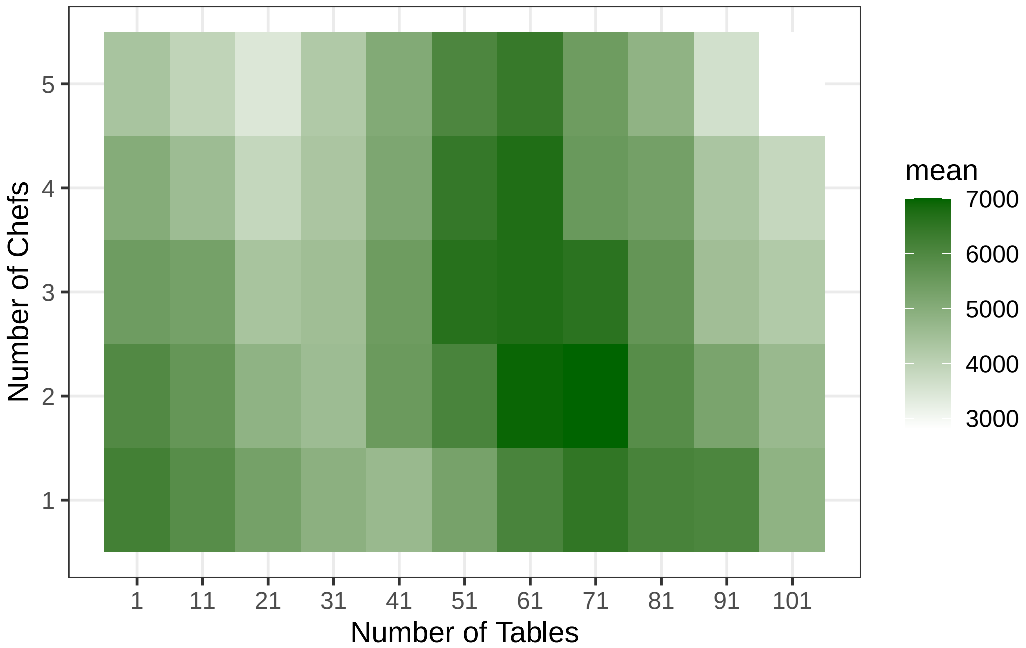
Arrival Times



Maximizing Profits

Under this scenario, how can we maximize profits?

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




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

```
# A tibble: 10 x 4
# Groups:   num_chefs [5]
   num_chefs num_tables mean_profit    sd
       <dbl>      <dbl>       <dbl> <dbl>
 1         2         71       7009. 1171.
 2         2         61       6948.  978.
 3         4         61       6735  1041.
 4         3         61       6730  1088.
 5         3         51       6652.  491.
 6         3         71       6595   868.
 7         1         71       6504.  462.
 8         4         51       6448.  567.
 9         5         61       6419. 1131.
10         1          1       6225   474.
```

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

```
# A tibble: 3 x 5
```

```
# Groups:   num_chefs [2]
  num_chefs num_tables profit mean_waiting_time    sd
      <dbl>      <dbl>  <dbl>             <dbl> <dbl>
1         2         71  7009.              3.63  1.93
2         2         61  6948.             10.5   3.72
3         4         61  6735               5.32  2.54
```

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

```
# A tibble: 3 x 5
# Groups:   num_chefs [2]
  num_chefs num_tables profit mean_long_waits    sd
      <dbl>      <dbl>  <dbl>           <dbl> <dbl>
1         2         71  7009.            1.75  3.68
2         2         61  6948.           38.3  25.3
3         4         61  6735             6.3  12.3
```

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

```
# A tibble: 3 x 5
# Groups:   num_chefs [2]
  num_chefs num_tables profit mean_queue_length    sd
      <dbl>      <dbl>  <dbl>             <dbl> <dbl>
1         2         71  7009.              1.43 0.829
2         2         61  6948.              4.13 1.75
3         4         61  6735               2.09 1.12
```

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

```
# A tibble: 3 x 5
# Groups:   num_chefs [2]
  num_chefs num_tables profit mean_max_queue    sd
      <dbl>      <dbl>  <dbl>          <dbl> <dbl>
1         2         71  7009.           17.6  5.63
2         2         61  6948.           28.4  5.48
3         4         61  6735            19.8  6.12
```

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

```
# A tibble: 3 x 5
# Groups:   num_chefs [2]
  num_chefs num_tables profit mean_occupied    sd
      <dbl>      <dbl>  <dbl>         <dbl> <dbl>
1         2         71  7009.          36.1  2.92
2         2         61  6948.          35.4  3.60
3         4         61  6735           30.5  2.56
```