By the Bayes (STA240 Project)

Noah Obuya & Tamya Davidson

2024-05-01

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.4.4     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

Proposal

Introduction

Welcome to By the Bayes! A restaurant serving food straight from the Louisiana Bayou. What makes our restaurant special is that every step of our process is backed by probabilistic modeling and inference. It's in our name. Today we'll be taking you through the customer's queuing experience in two different scenarios, providing simulations and explanations to back all of our claims.

Analysis Plan

Scenario One Our first scenario models simple setting with one dining table and one chef, with operating hours 10am - 10pm. Suppose customers arrive according to a Poisson process with a rate of $A = 5$ per hour.

$$\lambda_A = 5 \text{ per hour, } 40 \text{ total customers from open to close}$$

Once a customer arrives, their total service time (ordering, cooking and eating) can be modeled by an exponential distribution with rate $S = 6$.

$$\lambda_S = 6$$

To get a sense of the customer queuing experience, we will run multiple simulations of our exponential distribution. The simulation will model the time it takes to service one customer. Essentially, once one customer comes in, we want to know how long it takes for another customer to be serviced. We only have

one table and one chef, so we are only able to handle one customer at a time. We will build upon the exponential distribution simulation to create an algorithm that lets us know whether a not a table is ready for a customer.

To analyze the customer's queueing experience, set the average number of customers in the system = (L), the average number of customers waiting in line = (Lq), the average time a customer spends in the system = (W), and the average time a customer spends waiting in line = (Wq).

(L):represents the average number of customers, including those being served and those waiting in line. In this scenario, we can use the formula for an M/M/1 queue to calculate L: L= a/ − a

Scenario Two

References

Lab 10 Rossetti, R. (2024, April 4). Sakai.duke.edu. https://sakai.duke.edu/access/content/group/2e691ccc-405a-4982-bf81-42137192402e/Labs/lab10_sol.html

Simulation of Exponential Distribution using R Hernández, C. (2020, September 29). Simulation of Exponential Distribution using R. RPubs. https://rpubs.com/carlosehernandezr/Exponential-Distribution#:~:text=The%20exponential%20distribution%20can%20be,deviation%20is%20also%201%2Flambda

Analysis

Scenario One

```
lambda_s1 <- 6

x <- seq(0, 1, length.out = 1000)
y <- dexp(x, rate = lambda_s1)

meanwaitingtime <- data.frame(x = x, y = y)

p <- ggplot(meanwaitingtime, aes(x = x, y = y)) +
  geom_line() +
  geom_vline(aes(xintercept = 1/lambda_s1), linetype = "dashed", color = "red") +
  annotate("text", x = 1/lambda_s1, y = max(meanwaitingtime$y)/2, label = "Mean waiting time", hjust =
  labs(x = "Waiting time", y = "Density", title = "Exponential distribution with   = 6")
```

Part One - The Mean Figure 1 allows us to explain our mean wait time. Given the PDF (Probability Density Function) of our model, we see that the average wait time is 1/6 of an hour, or 10 minutes. This can also be described by

$$1/\lambda$$

, once again giving us 1/6 of an hour.

Part Two - Rudimentary Exponential Simulation accounting for a Que For our first model, we decided to look at the start and end times of each customer's service. The weight time is calculated by the difference between their service start time and their service end time.

```
lambda_A <- 5  # Arrival rate per hour
lambda_S <- 6  # Service rate per hour
num_customers <- 1000
num_simulations <- 3

# Function to run a single simulation and return the average wait time
run_simulation <- function() {
  arrival_times <- cumsum(rexp(num_customers, rate=lambda_A))
  service_start_times <- pmax(arrival_times, cummax(arrival_times[-1] + rexp(num_customers-1, rate=lambd
  service_end_times <- service_start_times + rexp(num_customers, rate=lambda_S)
  wait_times <- service_start_times - arrival_times
  return(mean(wait_times))
}

# Run simulations and calculate average wait times
average_wait_times <- replicate(num_simulations, run_simulation())
```

```
## Warning in pmax(arrival_times, cummax(arrival_times[-1] + rexp(num_customers -
## : an argument will be fractionally recycled

## Warning in pmax(arrival_times, cummax(arrival_times[-1] + rexp(num_customers -
## : an argument will be fractionally recycled

## Warning in pmax(arrival_times, cummax(arrival_times[-1] + rexp(num_customers -
## : an argument will be fractionally recycled
```

```
# Output average wait times for each run
cat("Average Wait Times for Each Run:", average_wait_times, "\n")
```

```
## Average Wait Times for Each Run: 0.4301262 0.4176693 0.4275473
```

The code above attempts to simulate the queueing process of customers at a dining establishment where arrival and service times follow Poisson and exponential distributions, respectively. The system is simulated for 1000 customers, and the simulation is repeated 3 times. For each simulation, arrival times are generated according to a Poisson process, and service times are modeled by an exponential distribution. The simulation calculates the start and end times of each customer's service and computes the wait time for each customer as the difference between their service start time and arrival time. The average wait time for each simulation run is then calculated and displayed.

However, average waiting time does not take into account if the customer comes in and has to wait for someone else, so the first model describes the customer coming in and having to wait for someone else, but this did not take into account the customers arrival time. Thus the next step in our analysis was to create a model that allowed us to take into consideration when the customer came into the restaurant (the arrival time).

Part Three - Fleshed Out Exponential Simulation accounting for a Que

```r
lambda_A <- 5  # Arrival rate per hour
lambda_S <- 6  # Service rate per hour
num_customers <- 1000
num_simulations <- 3

# Function to run a single simulation and return the average wait time
run_simulation <- function() {
  arrival_times <- cumsum(rexp(num_customers, rate=lambda_A))
  service_start_times <- numeric(num_customers)
  service_end_times <- numeric(num_customers)

  # Initialize the service times
  service_start_times[1] <- arrival_times[1]
  service_end_times[1] <- service_start_times[1] + rexp(1, rate=lambda_S)

  # Calculate service times for each subsequent customer
  for (i in 2:num_customers) {
    service_start_times[i] <- max(arrival_times[i], service_end_times[i-1])
    service_end_times[i] <- service_start_times[i] + rexp(1, rate=lambda_S)
  }

  wait_times <- service_start_times - arrival_times
  return(mean(wait_times))
}

# Run simulations and calculate average wait times
average_wait_times <- replicate(num_simulations, run_simulation())

# Output average wait times for each run
cat("Average Wait Times for Each Run:", average_wait_times, "\n")
```

```
## Average Wait Times for Each Run: 0.6468674 0.6404902 0.7288924
```

In part two, we attempted to calculate the start time of each customer's service as the maximum of their arrival time or the end time of the previous customer's service. Model one did not correctly account for the situation where the first customer's service time determines the second customer's potential wait, and it incorrectly handles the alignment of times for all customers. The service rate ,mu, is faster than the arrival rate lambda, which would generally suggest a system that doesn't build up a long queue. However, the theoretical waiting time still suggests a significant wait because, while the queue processes quickly, the relative closeness of the rates means each customer often has to wait for the previous one to finish. By increasing the number of customers and simulations, we got a more accurate representation of the theoretical values, averaging out the anomalies and fluctuations that were do to randomness. Model two also handled sequential dependency. Each customer's service start time correctly depends on both their arrival time and the exact end time of the previous customer's service. This ensures that the simulation accurately represents a queue where each customer waits if and only if the previous customer has not finished. The loop is also initialized correctly, as it sets up the first customer and then iterates through the rest, ensuring that each customer is handled accurately without index misalignment.

Scenario Two

In Scenario 2, we used the queueing model package from R. The package allowed us to calculate performance measures such as the average number of customers in the system, the average waiting time, the utilization of

servers, and the probability of having a certain number of customers in the system. By simulating different scenarios with varying numbers of chefs (L), we can determine the optimal number of chefs to minimize wait times and maximize profits. The package also allowed us to evaluate the impact of different factors, such as customer arrival rates and service times, on the restaurant's performance.

```
library(queueing)

simulate_day <- function(L) {
  lambda_A <- 10  # Arrival rate (customers per hour)
  num_tables <- 5  # Number of dining tables
  lambda_S <- 3 * L  # Service rate per hour based on number of chefs

  # M/M/c model where c = number of tables
  # and the service rate is adjusted based on the number of chefs
  Q_mm_c <- NewInput.MMC(lambda = lambda_A, mu = lambda_S, c = num_tables, n = 1000)
  result <- QueueingModel(Q_mm_c)

  # Extract average wait time in the queue (time spent waiting for a table)
  avg_wait_time <- result$Lq / lambda_A * 60  # Convert hours to minutes

  return(list(avg_wait_time = avg_wait_time, utilization = result$U))
}

# Testing with different number of chefs
simulate_day(2)
```

```
## $avg_wait_time
## [1] 0.09082982
##
## $utilization
## NULL
```

```
simulate_day(1)
```

```
## $avg_wait_time
## [1] 3.920031
##
## $utilization
## NULL
```

The model above uses a queueing system for our restaurant with a varying number of chefs LL. It uses the M/M/c queueing model, where:

- A A is the arrival rate of customers (10 customers per hour),
- **num_tables** is the number of dining tables (5 tables),
- S S is the service rate per hour, which is calculated as 3×L3×L based on the number of chefs.

  By simulating the queueing system with different numbers of chefs, the model helps determine the optimal number of chefs to minimize customer wait times and maximize profits.

```r
library(queueing)

# Function to simulate one day's operation and calculate profits
simulate_day <- function(L) {
  lambda_A <- 10  # Arrival rate (customers per hour)
  num_tables <- 5  # Number of dining tables
  lambda_S <- 3 * L  # Service rate per hour based on number of chefs
  opening_hour <- 10  # Restaurant opening hour
  closing_hour <- 22  # Restaurant closing hour

  # Using M/M/c queue model
  Q_mm_c <- NewInput.MMC(lambda = lambda_A, mu = lambda_S, c = num_tables, n = 1000)
  result <- QueueingModel(Q_mm_c)

  # Calculate hourly revenue based on the minimum of customer arrivals or table capacity
  hourly_revenue <- min(lambda_A, lambda_S * num_tables) * 50  # $50 per customer
  daily_revenue <- hourly_revenue * (closing_hour - opening_hour)  # Calculate hours of operation

  # Calculate daily wages for chefs
  daily_wages <- L * 40 * (closing_hour - opening_hour)  # $40 per hour per chef

  # Calculate profits
  profits <- daily_revenue - daily_wages

  # Get average wait time in minutes
  avg_wait_time <- result$Lq / lambda_A * 60

  return(list(profits = profits, avg_wait_time = avg_wait_time, revenue = daily_revenue, wages = daily_w
}

# Function to find optimal number of chefs
optimize_chefs <- function(max_chefs = 10) {
  results <- data.frame(Chefs = integer(), Profits = numeric(), AvgWaitTime = numeric(), Revenue = numer

  for (L in 1:max_chefs) {
    sim_results <- simulate_day(L)
    results[nrow(results) + 1, ] <- c(L, sim_results$profits, sim_results$avg_wait_time, sim_results$rev
  }

  optimal_chefs <- results[which.max(results$Profits), ]
  return(list(optimal = optimal_chefs, details = results))
}

# Execute the optimization function and print results
optimization_results <- optimize_chefs()
print(optimization_results$details)
```

```
##    Chefs Profits  AvgWaitTime Revenue Wages
## 1      1    5520 3.920031e+00    6000   480
## 2      2    5040 9.082982e-02    6000   960
## 3      3    4560 1.023647e-02    6000  1440
## 4      4    4080 2.095733e-03    6000  1920
## 5      5    3600 6.000825e-04    6000  2400
```

```
## 6      6    3120 2.134974e-04    6000  2880
## 7      7    2640 8.847236e-05    6000  3360
## 8      8    2160 4.105378e-05    6000  3840
## 9      9    1680 2.078810e-05    6000  4320
## 10    10    1200 1.128324e-05    6000  4800
```

```
cat("Optimal number of chefs for maximum profits:", optimization_results$optimal$Chefs, "\n")
```

```
## Optimal number of chefs for maximum profits: 1
```

This model simulates the daily operation of a restaurant to determine the optimal number of chefs that maximize profits while considering customer wait times. By varying the number of chefs, the model calculates the restaurant's revenue, chef wages, and profits for each scenario. It uses the M/M/c queuing model to simulate customer arrivals, service times, and queue lengths. The model then identifies the number of chefs that results in the highest profits.

Discussions & Limitations

Conclusion