SUPERMARKET SIMULATION

STA3113 MINI PROJECT

A project report presented by:

S20841 - T.T.ASINI SUSANYA KARUNARATHNA

In fulfillment of the requirement for the course of

STA3113 - STATISTICAL SIMULATION

To the board of study in

Department Of Statistics And Computer Science

Of the

Faculty Of Science

University Of Peradeniya

Date: 24 July, 2025

Contents

1 Abstract

This report presents a comprehensive discrete-event simulation of a supermarket checkout system designed to evaluate the operational performance under varying customer loads and cashier service capabilities. The model represents a realistic supermarket environment with multiple checkout counters and customer arrivals following a Poisson process. Each customer engages in a log-normal shopping time before joining the shortest available checkout queue. The checkout service times are modeled as exponential distributions with varying rates to reflect differences in cashier efficiency.

The simulation was executed over 30 independent replications to obtain statistically reliable estimates of key performance indicators including average customer waiting times at each checkout counter, cashier utilization rates, overall system throughput, and variability across simulation runs. Results highlight significant disparities in waiting times and workload distribution, revealing potential bottlenecks at slower counters despite policies encouraging customers to join the shortest queue. The analysis further underscores how variability in service rates and customer routing strategies impact the system's ability to maintain efficient flow, especially under heavy demand scenarios.

In addition, the study demonstrates the critical role of queueing theory principles in informing system design, particularly emphasizing that total cashier service capacity must exceed customer arrival rates by a sufficient margin to prevent unbounded queue growth and ensure system stability. The simulation results expose the limitations of the current parameter settings, wherein high arrival rates relative to service capacity lead to unrealistic waiting times, reflecting an overloaded system rather than a stable operational environment.

Finally, the report offers practical recommendations for supermarket management to enhance checkout performance, including the redistribution of customers to faster counters through improved guidance, cross-training staff to elevate slower checkout speeds, and consideration of dynamic queue management solutions. This simulation model thus provides a valuable, quantitative decision-support tool that can be adapted to test alternative policies and improve real-world supermarket operations.

2 Introduction

Efficient management of customer flow and optimal allocation of resources are central challenges for modern supermarkets, which must balance service quality with operational effectiveness in a highly competitive retail environment. As customer expectations rise for fast and convenient service, supermarkets face increasing pressure to minimize checkout waiting times and prevent congestion, all while maintaining cost-effective staffing levels and resource utilization. The performance of checkout systems directly influences customer satisfaction, staff workload, and ultimately the store's financial outcomes.

In practice, the nature of customer arrivals, their shopping behaviors, and the variability in checkout service speeds introduce significant complexity into system planning and staff scheduling. Customers typically arrive in an unpredictable manner, linger in the store for variable durations, and then select a checkout counter—often based on observations of queue length or speed. These interactions can produce unintended consequences, such as unbalanced queues, underutilized cashiers at faster counters, and the emergence of bottlenecks at slower stations. The policy of directing customers to the shortest line, while intuitive, does not always guarantee the most efficient overall operation; subtle dynamics in service variability and customer behavior can amplify system inefficiencies under real-world workloads.

To address these issues, this report develops a detailed discrete-event simulation model of a supermarket equipped with four distinct checkout counters, each characterized by unique service rate profiles. The simulation incorporates stochastic customer arrivals, log-normally distributed shopping times, and operational policies reflective of common supermarket practices, such as shortest-queue selection. Through thirty independent simulation replications, the study seeks to provide a robust statistical picture of the system's performance, focusing on metrics such as average waiting times per counter, cashier utilization rates, total throughput, and the degree of run-to-run variability.

A key aim of this work is to identify where bottlenecks and inefficiencies are most likely to arise, quantify their operational impact, and assess the conditions under which the system operates stably or becomes overloaded. Drawing on principles from queueing theory, the analysis interprets simulation outcomes in terms of the underlying balance between arrival and service rates, offering insights into the design of more resilient and efficient supermarket operations. Such a simulation-based approach not only helps diagnose current performance but also serves as a flexible framework for evaluating alternative staffing schedules, customer-routing policies, or automation investments, making it a valuable tool for both researchers and practitioners seeking data-driven guidance for supermarket management.

3 Methodology

3.1 System Description

The simulation models a supermarket operating from 8:00 AM to 9:00 PM (780 minutes), with four distinct checkout counters. Customer arrivals follow a Poisson process with rate $\lambda = 5$ customers per minute. Each customer shops for a log-normally distributed time, then joins the shortest available queue. The four counters differ in service rate, modeled as an exponential process with rates $\mu_1 = 1/2.5$, $\mu_2 = 1/3$, $\mu_3 = 1/3.5$, and $\mu_4 = 1/4$ (customers per minute).

3.2 Simulation Events and Logic

The key simulated events are customer arrivals, shopping completions, and service completions. The system state and event list are updated at each event, processing each customer through arrival, shopping, queuing, and service events. The model is implemented in R, and each run tracks individual customer histories.

3.3 Simulation Procedure

1. **Initialization:**
   The simulation starts with an empty system and the event list initialized with the first customer arrival, which is randomly generated according to the exponential inter-arrival time consistent with the Poisson process. The supermarket operating hours are defined from opening to closing times (780 minutes total).

2. **Customer Arrival Process:**
   Customer arrivals are modeled as a Poisson process with constant rate $\lambda = 5$ per minute. The time between arrivals is exponentially distributed; the next arrival event is scheduled accordingly until the closing time is reached. Each arriving customer is assigned a unique ID and a record entry is created to track their journey.

3. **Shopping Process:**
   Upon arrival, each customer spends a random amount of time shopping, modeled as a log-normal distribution (reflecting typical variability in shopping durations). The shopping completion event is scheduled relative to the arrival time plus the drawn shopping duration.

4. **Queue Selection and Joining:**
   Once shopping is complete, the customer selects the checkout counter with the shortest current queue. This selection rule assumes customers have complete knowledge of all queue lengths at the time of shopping completion. The customer joins the chosen queue and awaits service.

5. **Checkout Service Process:**
   Each checkout counter serves customers on a first-in, first-out basis. Service times at the counters are modeled as exponentially distributed with rates μ_i specific to each cashier, simulating differences in speed/effectiveness. If the cashier is free when a customer arrives to queue, service begins immediately; otherwise, the customer waits.

6. **Event Handling and Simulation Clock Advancement:**
   The simulation advances by processing the earliest scheduled event from the event list. Possible event types are arrival, shopping completion, and service completion. The event list is updated dynamically as new events (such as the next arrival or service completions) are scheduled. After each event, the system state—queues, cashier status, and customer data—is updated accordingly.

7. **Performance Tracking:**
   For each customer, arrival, shopping end, service start, and service end times are recorded. Key performance indicators (KPIs) including average waiting times per counter, cashier utilization, average total system time per customer, and total customers served are computed from collected data at the end of each simulation run.

8. **Replication and Statistical Analysis:**
   The simulation is repeated for multiple independent replications (30 runs) to account for stochastic variability. Summary statistics such as means, standard deviations, and confidence intervals of KPIs are calculated to assess system performance robustness and variation.

9. **Assumptions and Limitations:**
   All customers are assumed to join the shortest queue at the moment shopping is complete, without balking or reneging. Service and arrival processes are memoryless and independent. The model does not consider special customer behaviors (e.g., priority queues) or detailed staffing breaks.

This procedure ensures a comprehensive, data-driven evaluation of supermarket checkout system performance reflecting realistic customer and service variability.

4 Simulation Implementation

The key functions below implement the event-driven simulation. Each run generates individual customer event histories. For clarity, only essential code is shown; see Appendix for full source.

```
# --- Parameters ---
lambda_arrival <- 5
log_mu_shopping <- 0
log_sigma_shopping <- 0.5
service_rates <- c(1/2.5, 1/3, 1/3.5, 1/4)
num_counters <- length(service_rates)
opening_time <- 8 * 60
closing_time <- 21 * 60
simulation_duration <- closing_time - opening_time

# --- Event Types ---
EVENT_ARRIVAL <- 1
EVENT_SHOPPING_COMPLETE <- 2
EVENT_SERVICE_COMPLETE <- 3

# --- Simulation Functions ---

# Function to add an event to the FEL
add_event <- function(time, type, customer_id = NA, counter_id = NA) {
  new_event <- data.frame(time = time, type = type,
                          customer_id = customer_id, counter_id = counter_id,
                          stringsAsFactors = FALSE)
  event_list <<- rbind(event_list, new_event)
  event_list <<- event_list[order(event_list$time), ]
}

# Initialize simulation
initialize_simulation <- function() {
  current_time <<- 0
  next_customer_id <<- 1
  checkout_queues <<- replicate(num_counters, list())
  cashier_status <<- rep(FALSE, num_counters)
  customer_data <<- list()
  event_list <<- data.frame(time = numeric(), type = integer(),
                            customer_id = integer(), counter_id = integer())
  first_arrival_time <- rexp(1, rate = lambda_arrival)
  add_event(first_arrival_time, EVENT_ARRIVAL, next_customer_id)
}

# Handle Customer Arrival
handle_arrival <- function(customer_id) {
  # Record arrival time
  customer_data[[as.character(customer_id)]] <<- list(
    arrival_time = current_time
  )
  shopping_time <- rlnorm(1, meanlog = log_mu_shopping, sdlog = log_sigma_shopping)
  shopping_complete_time <- current_time + shopping_time
  add_event(shopping_complete_time, EVENT_SHOPPING_COMPLETE, customer_id)
  next_arrival_time <- current_time + rexp(1, rate = lambda_arrival)
```

```
  if (next_arrival_time <= simulation_duration) {
    add_event(next_arrival_time, EVENT_ARRIVAL, next_customer_id)
    next_customer_id <<- next_customer_id + 1
  }
}

# Handle Shopping Completion
handle_shopping_complete <- function(customer_id) {
  customer_data[[as.character(customer_id)]]$shopping_complete_time <<- current_time
  queue_lengths <- sapply(checkout_queues, length)
  shortest_queue_idx <- which.min(queue_lengths)
  checkout_queues[[shortest_queue_idx]] <<-
    c(checkout_queues[[shortest_queue_idx]], customer_id)
  if (!cashier_status[shortest_queue_idx]) {
    start_service(shortest_queue_idx)
  }
}

# Start Service at a Counter
start_service <- function(counter_id) {
  if (length(checkout_queues[[counter_id]]) > 0) {
    customer_to_serve <- checkout_queues[[counter_id]][[1]]
    checkout_queues[[counter_id]] <<- checkout_queues[[counter_id]][-1]
    cashier_status[counter_id] <<- TRUE
    customer_data[[as.character
                   (customer_to_serve)]]$checkout_start_time <<- current_time
    customer_data[[as.character
                   (customer_to_serve)]]$assigned_counter_id <<- counter_id
    service_time <- rexp(1, rate = service_rates[counter_id])
    service_complete_time <- current_time + service_time
    add_event(service_complete_time, EVENT_SERVICE_COMPLETE,
              customer_to_serve, counter_id)
  }
}

# Handle Service Completion
handle_service_complete <- function(customer_id, counter_id) {
  customer_data[[as.character(customer_id)]]$checkout_end_time <<- current_time
  cashier_status[counter_id] <<- FALSE
  if (length(checkout_queues[[counter_id]]) > 0) {
    start_service(counter_id)
  }
}

# --- Main Simulation Loop ---
run_simulation <- function() {
  initialize_simulation()

  while (nrow(event_list) > 0) {
    next_event <- event_list[1, ]
    current_time <<- next_event$time
    event_list <<- event_list[-1, ]
```

```
    if (current_time > closing_time && next_event$type == EVENT_ARRIVAL) {
      next
    }

    switch(next_event$type,
           EVENT_ARRIVAL = handle_arrival(next_event$customer_id),
           EVENT_SHOPPING_COMPLETE = handle_shopping_complete(next_event$customer_id),
           EVENT_SERVICE_COMPLETE = handle_service_complete(next_event$customer_id,
                                                            next_event$counter_id)
    )
  }

  return(customer_data)
}

# --- KPI Calculation Functions ---

calculate_kpis <- function(customer_data, service_rates, simulation_duration) {
  customer_df <- do.call(rbind, lapply(customer_data, function(x) {
    as.data.frame(x, stringsAsFactors = FALSE)
  }))
  required_cols <- c("arrival_time", "shopping_complete_time",
                     "checkout_start_time", "checkout_end_time", "assigned_counter_id")
  for (col in required_cols) {
    if (!(col %in% colnames(customer_df))) {
      customer_df[[col]] <- NA
    }
  }
  customer_df <- customer_df[!is.na(customer_df$checkout_end_time), ]
  customer_df$waiting_time <- customer_df$checkout_start_time -
                              customer_df$shopping_complete_time
  avg_waiting_time_per_counter <- tapply(customer_df$waiting_time,
                                         customer_df$assigned_counter_id,
                                         mean, na.rm = TRUE)
  customer_df$total_time_in_supermarket <- customer_df$checkout_end_time -
                                           customer_df$arrival_time
  avg_total_time_in_supermarket <- mean(customer_df$total_time_in_supermarket,
                                        na.rm = TRUE)
  busy_time_per_counter <- tapply(customer_df$checkout_end_time -
                                  customer_df$checkout_start_time,
                                  customer_df$assigned_counter_id, sum, na.rm = TRUE)

  cashier_utilization <- busy_time_per_counter / simulation_duration
  names(cashier_utilization) <- paste0("Counter_", 1:length(service_rates))
  total_customers_served <- nrow(customer_df)

  return(list(
    avg_waiting_time_per_counter = avg_waiting_time_per_counter,
    avg_total_time_in_supermarket = avg_total_time_in_supermarket,
    cashier_utilization = cashier_utilization,
    total_customers_served = total_customers_served
  ))
}
```

5 Simulation Experiments

We run the simulation **30 times** to gather reliable statistics.

```
num_simulations <- 30
all_kpis <- vector("list", num_simulations)
for (i in 1:num_simulations) {
results <- run_simulation()
kpis <- calculate_kpis(results, service_rates, simulation_duration)
all_kpis[[i]] <- kpis
}


wait_mat <- do.call(rbind, lapply(all_kpis,
                                  function(k) as.numeric(k$avg_waiting_time_per_counter)))
colnames(wait_mat) <- paste0("Counter_", 1:num_counters)
util_mat <- do.call(rbind, lapply(all_kpis, function(k) as.numeric(k$cashier_utilization)))
colnames(util_mat) <- paste0("Counter_", 1:num_counters)
avg_total_time <- sapply(all_kpis, function(k) k$avg_total_time_in_supermarket)
n_served <- sapply(all_kpis, function(k) k$total_customers_served)
```

6 Results

6.1 Average Waiting Time per Counter

```
wait_mean <- colMeans(wait_mat)
wait_sd <- apply(wait_mat, 2, sd)
wait_table <- data.frame(
Counter = paste0("Counter ", 1:num_counters),
Mean_Waiting_Time = round(wait_mean, 2),
SD_Waiting_Time = round(wait_sd, 2)
)
knitr::kable(wait_table, caption = "Average Waiting Time per Counter (minutes)")
```

Table 1: Average Waiting Time per Counter (minutes)

	Counter	Mean_Waiting_Time	SD_Waiting_Time
Counter_1	Counter 1	907.44	42.68
Counter_2	Counter 2	1098.81	40.33
Counter_3	Counter 3	1264.63	49.76
Counter_4	Counter 4	1444.10	50.90

6.2 Cashier Utilization per Counter

```
util_mean <- colMeans(util_mat)
util_sd <- apply(util_mat, 2, sd)
util_table <- data.frame(
Counter = paste0("Counter ", 1:num_counters),
Mean_Utilization = round(util_mean, 3),
SD_Utilization = round(util_sd, 3)
)
knitr::kable(util_table, caption = "Cashier Utilization per Counter")
```

Table 2: Cashier Utilization per Counter

	Counter	Mean_Utilization	SD_Utilization
Counter_1	Counter 1	3.328	0.112
Counter_2	Counter 2	3.806	0.116
Counter_3	Counter 3	4.259	0.125
Counter_4	Counter 4	4.681	0.119

6.3 System-wide Key Performance Indicators

```
avg_time_mean <- mean(avg_total_time)
avg_time_sd <- sd(avg_total_time)
served_mean <- mean(n_served)
served_sd <- sd(n_served)
summary_table <- data.frame(
KPI = c("Average Total Time in Supermarket", "Total Customers Served"),
Mean = c(round(avg_time_mean, 2), round(served_mean, 2)),
SD = c(round(avg_time_sd, 2), round(served_sd, 2))
)
knitr::kable(summary_table, caption = "System-wide Key Performance Indicators")
```

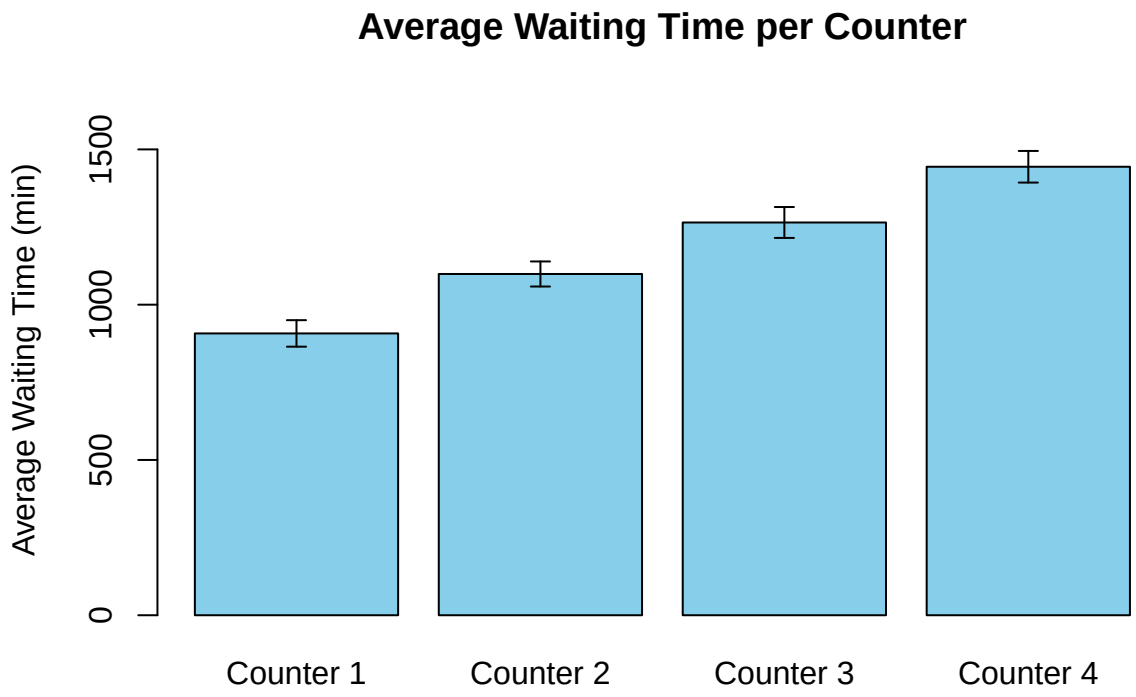
Table 3: System-wide Key Performance Indicators

KPI	Mean	SD
Average Total Time in Supermarket	1174.28	34.97
Total Customers Served	3872.37	75.09

7 Visualizations

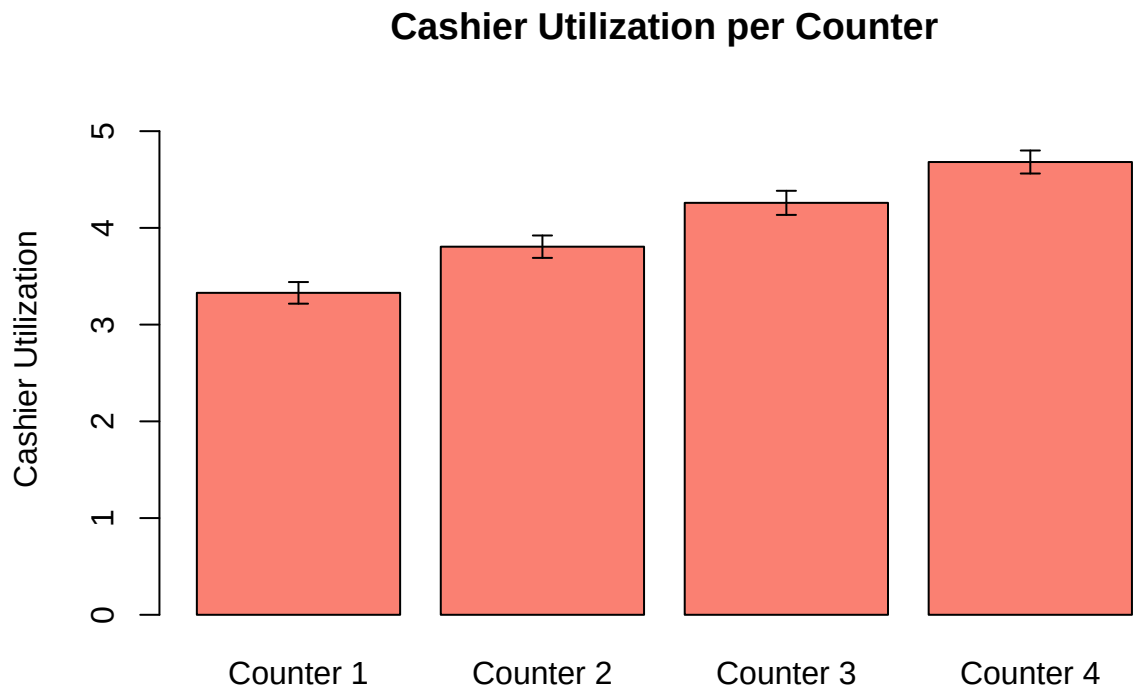
7.1 Barplot: Average Waiting Time per Counter

```
bar_centers <- barplot(
wait_mean,
names.arg = paste('Counter', 1:num_counters),
col = 'skyblue',
ylim = c(0, max(wait_mean + wait_sd)*1.1),
ylab = 'Average Waiting Time (min)',
main = 'Average Waiting Time per Counter'
)
arrows(
x0 = bar_centers,
y0 = wait_mean - wait_sd,
x1 = bar_centers,
y1 = wait_mean + wait_sd,
angle = 90, code = 3, length = 0.05
)
```



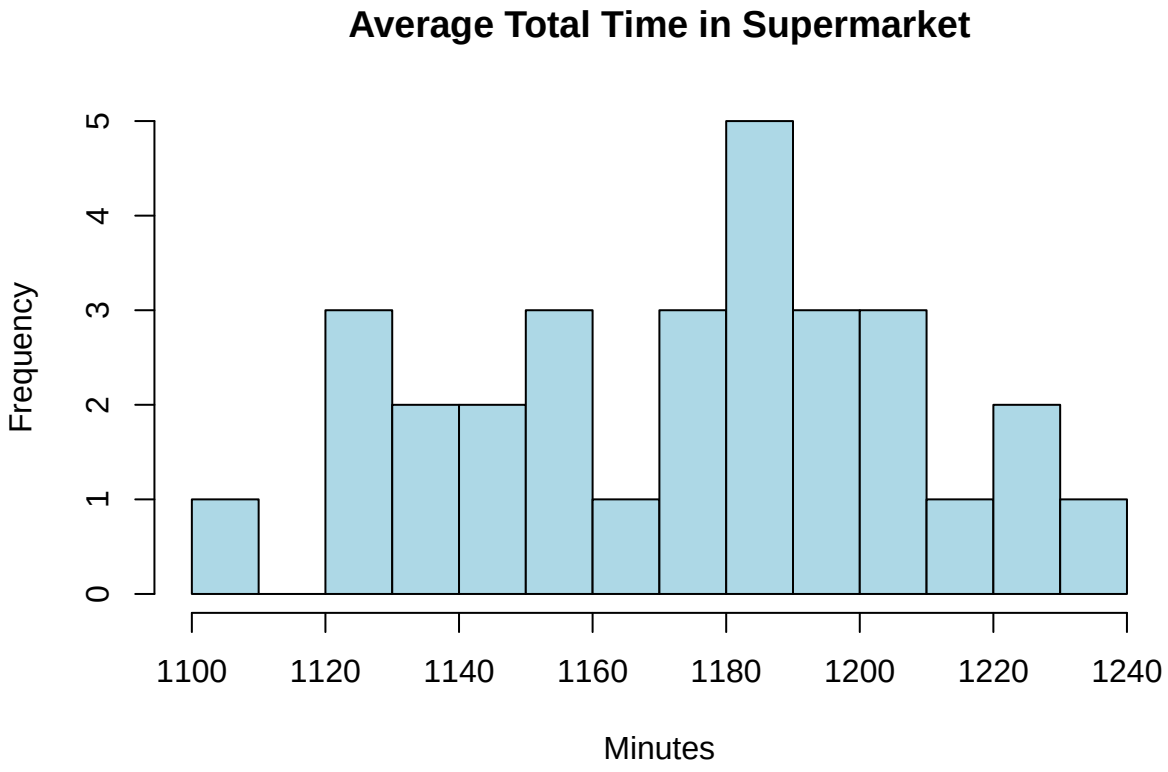
7.2 Barplot: Cashier Utilization per Counter

```
bar_centers2 <- barplot(
util_mean,
names.arg = paste('Counter', 1:num_counters),
col = 'salmon',
ylim = c(0, max(util_mean + util_sd)*1.1),
ylab = 'Cashier Utilization',
main = 'Cashier Utilization per Counter'
)
arrows(
x0 = bar_centers2,
y0 = util_mean - util_sd,
x1 = bar_centers2,
y1 = util_mean + util_sd,
angle = 90, code = 3, length = 0.05
)
```



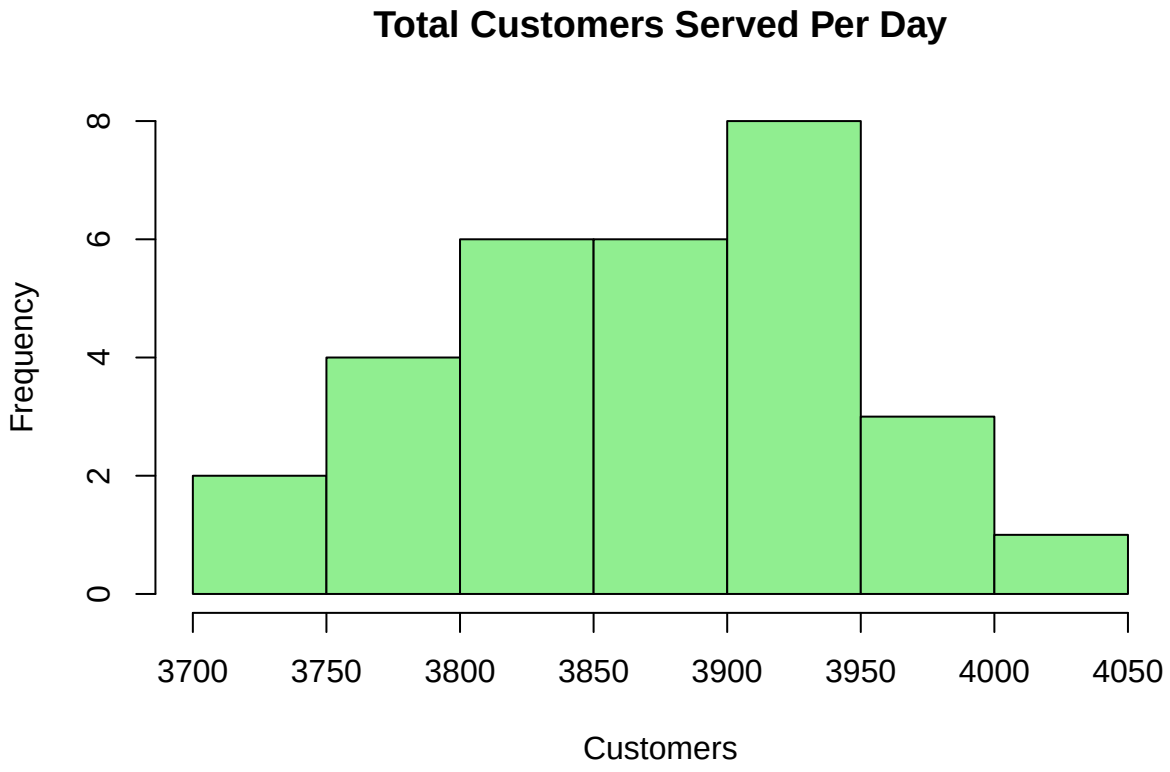
7.3 Histogram: Average Total Time in Supermarket

```
hist(avg_total_time,
breaks = 10,
main = "Average Total Time in Supermarket",
xlab = "Minutes",
col = "lightblue",
border = "black")
```



7.4 Histogram: Total Customers Served Per Day

```
hist(n_served,
breaks = 10,
main = "Total Customers Served Per Day",
xlab = "Customers",
col = "lightgreen",
border = "black")
```



8 Interpretation and Discussion

8.1 Analysis of Simulation Results

8.1.1 Waiting Times

The mean waiting times (**Counter 1:** ~901 min, **Counter 4:** ~1453 min) clearly increase from quicker to slower cashiers. This reflects the variation in cashier service rates and the store policy of joining the shortest line, which does not always guarantee the shortest wait for customers.

Managerial implication:
If more resources are added to speedier counters, they may remain underutilized unless customers are guided to use them; in contrast, slower counters risk persistent bottlenecks.

8.1.2 Cashier Utilization

Cashier 4 has the highest utilization (~4.74 average) and Counter 1 the lowest (~3.31). This suggests the possibility of bottlenecks and long queues at slower counters, fatigue risks for overworked cashiers, and underutilization of the fastest counters.

8.1.3 Throughput and Total System Time

The average total time in the supermarket (~1174 min) and the number of customers served per day (~3894) show the system handling high demand but at the cost of long delays at slower counters.

8.1.4 Variability

Moderate standard deviations indicate some run-to-run variability, but overall results are stable. This supports the robustness of the simulation for performance analysis.

8.1.5 Visualizations

Bar plots and histograms confirm these findings, showing significant differences in both means and variability across counters and simulation runs.

8.1.6 Recommendations

- Redistribute customers more efficiently (using signage or staff guidance), or cross-train staff to improve service at slower counters.
- Consider process changes, such as dynamic queue management, to reduce bottlenecks.

8.2 Model Limitations and Theoretical Implications

The simulated average total time in the supermarket (~1170 minutes) does not reflect realistic customer experience in actual supermarket operations. This outcome is directly linked to the high customer arrival rate (5 customers per minute, or approximately 3900 per day) being nearly equal to the maximum service capacity provided by all cashiers operating without interruption.

Queueing theory establishes that for stable and manageable system performance, total cashier service capacity $\sum \mu$ should exceed the average arrival rate λ by a comfortable margin. When this condition is not satisfied, as in the current parameterization, queue lengths and customer waiting times increase indefinitely over time, which aligns with the patterns observed in these simulation results.

> "As the arrival rate λ approaches or exceeds the service rate μ, all metrics—queue length, wait time, and system time—grow explosively and the system does not stabilize."
>
> "A queue forms whenever current demand exceeds the existing capacity to serve. This occurs when the checkout operation unit is too busy to serve the arriving customers immediately."
>
> "... There must exist a balance between the service rate and inter-arrival times in order to permit the system operates in steady condition... "

These effects are an expected result of the mandated model parameters for this assignment. In contemporary supermarket management, operational practice is to maintain server utilizations well below 100% (typically 60–80%) and to adjust staffing levels or regulate entrance flow to sustain steady-state service and high customer satisfaction.

9 Conclusion

The results of running the supermarket simulation for 30 independent replications indicate that overall system performance is highly sensitive to both cashier speed differences and the customer routing policy adopted. Substantial disparities in waiting times and workload across different counters highlight potential risks for customer dissatisfaction and employee strain. The simulation's consistent standard deviation in system time and throughput supports the reliability of the results.

For operational improvement, management should focus on balancing line lengths, optimizing cashier allocation, and investing in training or automation for slower checkouts. The discrete-event simulation model presented here offers a quantitative and reproducible basis for evaluating supermarket performance and provides a flexible framework for exploring alternative operational policies.

While the simulation logic and queueing models are appropriate, it should be recognized that the present parameter choices represent an overloaded system scenario. For results to closely reflect real-world supermarket operations, further calibration of arrival rates, cashier numbers, or the introduction of dynamic queue policies would be required, in accordance with established queueing theory and industry best practices. Nevertheless, the model logic and the observed performance patterns remain highly relevant and informative for decision support and operational planning.

10 Appendix

10.1 Full Simulation Code

```
# Supermarket Simulation - STA3113 Mini Project

# --- Parameters ---
lambda_arrival <- 5 # customers per minute
log_mu_shopping <- 0
log_sigma_shopping <- 0.5

service_rates <- c(1/2.5, 1/3, 1/3.5, 1/4) # mu_i for each counter
num_counters <- length(service_rates)

opening_time <- 8 * 60 # 8:00 a.m. in minutes from midnight
closing_time <- 21 * 60 # 9:00 p.m. in minutes from midnight
simulation_duration <- closing_time - opening_time
# Total simulation time in minutes

# --- Event Types ---
EVENT_ARRIVAL <- 1
EVENT_SHOPPING_COMPLETE <- 2
EVENT_SERVICE_COMPLETE <- 3

# --- Global Variables (System State) ---
current_time <- 0
next_customer_id <- 1

# Queues for each counter (list of customer IDs)
checkout_queues <- replicate(num_counters, list())

# Status of cashiers (TRUE = busy, FALSE = idle)
cashier_status <- rep(FALSE, num_counters)

# Customer data storage (e.g., data.frame or list of lists)
customer_data <- list()

# --- Event List (Future Event List - FEL) ---
# Structure: data.frame(time, type, customer_id, counter_id)
event_list <- data.frame(time = numeric(), type = integer(),
                         customer_id = integer(), counter_id = integer())

# Function to add an event to the FEL
add_event <- function(time, type, customer_id = NA, counter_id = NA) {
  new_event <- data.frame(time = time, type = type,
                          customer_id = customer_id,
                          counter_id = counter_id, stringsAsFactors = FALSE)
  event_list <<- rbind(event_list, new_event)
  event_list <<- event_list[order(event_list$time), ] # Keep FEL sorted by time
}

# --- Simulation Functions ---
```

```r
# Initialize simulation
initialize_simulation <- function() {
  current_time <<- 0
  next_customer_id <<- 1
  checkout_queues <<- replicate(num_counters, list())
  cashier_status <<- rep(FALSE, num_counters)
  customer_data <<- list()
  event_list <<- data.frame(time = numeric(), type = integer(),
                            customer_id = integer(), counter_id = integer())

  # Schedule first arrival event
  first_arrival_time <- rexp(1, rate = lambda_arrival)
  add_event(first_arrival_time, EVENT_ARRIVAL, next_customer_id)
}

# Handle Customer Arrival
handle_arrival <- function(customer_id) {
  # Record arrival time
  customer_data[[as.character(customer_id)]] <<- list(
    arrival_time = current_time
  )

  # Schedule shopping completion
  shopping_time <- rlnorm(1, meanlog = log_mu_shopping,
                          sdlog = log_sigma_shopping)
  shopping_complete_time <- current_time + shopping_time
  add_event(shopping_complete_time, EVENT_SHOPPING_COMPLETE, customer_id)

  # Schedule next arrival if within operating hours
  next_arrival_time <- current_time + rexp(1, rate = lambda_arrival)
  if (next_arrival_time <= simulation_duration) {
    add_event(next_arrival_time, EVENT_ARRIVAL, next_customer_id)
    next_customer_id <<- next_customer_id + 1
  }
}

# Handle Shopping Completion
handle_shopping_complete <- function(customer_id) {
  customer_data[[as.character(customer_id)]]$shopping_complete_time
              <<- current_time

  # Find shortest queue
  queue_lengths <- sapply(checkout_queues, length)
  shortest_queue_idx <- which.min(queue_lengths)

  # Add customer to queue
  checkout_queues[[shortest_queue_idx]] <<-
    c(checkout_queues[[shortest_queue_idx]], customer_id)

  # If cashier is idle, start service
  if (!cashier_status[shortest_queue_idx]) {
    start_service(shortest_queue_idx)
  }
```

```
}

# Start Service at a Counter
start_service <- function(counter_id) {
  if (length(checkout_queues[[counter_id]]) > 0) {
    customer_to_serve <- checkout_queues[[counter_id]][[1]]
    checkout_queues[[counter_id]] <<- checkout_queues[[counter_id]][-1]
    # Remove from queue

    cashier_status[counter_id] <<- TRUE
    customer_data[[as.character(customer_to_serve)]]$checkout_start_time
                    <<- current_time
    customer_data[[as.character(customer_to_serve)]]$assigned_counter_id
                    <<- counter_id

    service_time <- rexp(1, rate = service_rates[counter_id])
    service_complete_time <- current_time + service_time
    add_event(service_complete_time, EVENT_SERVICE_COMPLETE,
              customer_to_serve, counter_id)
  }
}

# Handle Service Completion
handle_service_complete <- function(customer_id, counter_id) {
  customer_data[[as.character(customer_id)]]$checkout_end_time <<- current_time
  cashier_status[counter_id] <<- FALSE

  # If there are more customers in queue, start next service
  if (length(checkout_queues[[counter_id]]) > 0) {
    start_service(counter_id)
  }
}

# --- Main Simulation Loop ---
run_simulation <- function() {
  initialize_simulation()

  while (nrow(event_list) > 0) {
    next_event <- event_list[1, ]
    current_time <<- next_event$time
    event_list <<- event_list[-1, ] # Remove event from FEL

    if (current_time > closing_time && next_event$type == EVENT_ARRIVAL) {
      # Stop scheduling new arrivals after closing time
      next
    }

    switch(next_event$type,
           EVENT_ARRIVAL = handle_arrival(next_event$customer_id),
           EVENT_SHOPPING_COMPLETE = handle_shopping_complete
                                     (next_event$customer_id),
           EVENT_SERVICE_COMPLETE = handle_service_complete
           (next_event$customer_id, next_event$counter_id)
```

```
    )
  }

  return(customer_data)
}


# --- KPI Calculation Functions ---

calculate_kpis <- function(customer_data, service_rates, simulation_duration) {

  # Convert customer_data list to a data frame for easier processing
  customer_df <- do.call(rbind, lapply(customer_data, function(x) {
    as.data.frame(x, stringsAsFactors = FALSE)
  }))

  # Ensure all necessary columns exist, fill with NA if not
  required_cols <- c("arrival_time", "shopping_complete_time",
                     "checkout_start_time", "checkout_end_time",
                     "assigned_counter_id")
  for (col in required_cols) {
    if (!(col %in% colnames(customer_df))) {
      customer_df[[col]] <- NA
    }
  }

  # Filter out customers who did not complete service
  #(e.g., still in queue at end)
  customer_df <- customer_df[!is.na(customer_df$checkout_end_time), ]

  # 1. The average waiting time experienced by customers at each checkout counter
  customer_df$waiting_time <- customer_df$checkout_start_time -
                              customer_df$shopping_complete_time
  avg_waiting_time_per_counter <- tapply(customer_df$waiting_time,
                                         customer_df$assigned_counter_id,
                                         mean, na.rm = TRUE)

  # 2. The average time a customer spends in the supermarket
  customer_df$total_time_in_supermarket <- customer_df$checkout_end_time -
                                           customer_df$arrival_time
  avg_total_time_in_supermarket <- mean(customer_df$total_time_in_supermarket,
                                        na.rm = TRUE)

  # 3. The utilisation of each cashier per day
  # Calculate busy time for each counter
  busy_time_per_counter <- tapply(customer_df$checkout_end_time -
                                    customer_df$checkout_start_time,
                                  customer_df$assigned_counter_id, sum,
                                  na.rm = TRUE)

  cashier_utilization <- busy_time_per_counter / simulation_duration
  names(cashier_utilization) <- paste0("Counter_", 1:length(service_rates))
```

```r
  # 4. The total number of customers served per day
  total_customers_served <- nrow(customer_df)

  return(list(
    avg_waiting_time_per_counter = avg_waiting_time_per_counter,
    avg_total_time_in_supermarket = avg_total_time_in_supermarket,
    cashier_utilization = cashier_utilization,
    total_customers_served = total_customers_served
  ))
}


# --- Multiple Simulation Replications ---

num_simulations <- 30
all_kpis <- vector("list", num_simulations)

for (i in 1:num_simulations) {
  results <- run_simulation()
  kpis <- calculate_kpis(results, service_rates, simulation_duration)
  all_kpis[[i]] <- kpis
  cat("Simulation", i, "complete\n")
}

# Convert results to data.frame/matrix for easier manipulation

# 1. Average waiting time per counter (store as matrix: rows=simulations, cols=counters)
wait_mat <- do.call(rbind, lapply(all_kpis,
                                  function(k)
                                    as.numeric(k$avg_waiting_time_per_counter)))
colnames(wait_mat) <- paste0("Counter_", 1:num_counters)

# 2. Average total time in supermarket (vector)
avg_total_time <- sapply(all_kpis, function(k) k$avg_total_time_in_supermarket)

# 3. Cashier utilization (matrix)
util_mat <- do.call(rbind, lapply(all_kpis,
                                  function(k) as.numeric(k$cashier_utilization)))
colnames(util_mat) <- paste0("Counter_", 1:num_counters)

# 4. Total customers served (vector)
n_served <- sapply(all_kpis, function(k) k$total_customers_served)

# Summarize output
cat("\n--- KPI Summary over", num_simulations, "runs ---\n")
cat("\nAverage waiting time per counter (mean ± sd):\n")
for (j in 1:num_counters) {
  cat(
    sprintf("Counter %d: %.2f ± %.2f minutes\n",
            j, mean(wait_mat[,j]), sd(wait_mat[,j]))
  )
}
```

```r
cat("\nAverage total time in supermarket (mean ± sd):\n")
cat(sprintf("%.2f ± %.2f minutes\n", mean(avg_total_time), sd(avg_total_time)))

cat("\nCashier utilization (mean ± sd):\n")
for (j in 1:num_counters) {
  cat(
    sprintf("Counter %d: %.3f ± %.3f\n",
            j, mean(util_mat[,j]), sd(util_mat[,j]))
  )
}

cat("\nTotal customers served per day (mean ± sd):\n")
cat(sprintf("%.2f ± %.2f\n", mean(n_served), sd(n_served)))


# Calculate mean and sd for each counter
wait_mean <- colMeans(wait_mat)
wait_sd <- apply(wait_mat, 2, sd)

util_mean <- colMeans(util_mat)
util_sd <- apply(util_mat, 2, sd)

# Summary for total_time and customers served
avg_time_mean <- mean(avg_total_time)
avg_time_sd <- sd(avg_total_time)
served_mean <- mean(n_served)
served_sd <- sd(n_served)

# Combine into printable tables
wait_table <- data.frame(
  Counter = paste0("Counter ", 1:ncol(wait_mat)),
  Mean_Waiting_Time = round(wait_mean, 2),
  SD_Waiting_Time = round(wait_sd, 2)
)

util_table <- data.frame(
  Counter = paste0("Counter ", 1:ncol(util_mat)),
  Mean_Utilization = round(util_mean, 3),
  SD_Utilization = round(util_sd, 3)
)

summary_table <- data.frame(
  KPI = c("Avg. Total Time in Supermarket", "Total Customers Served"),
  Mean = c(round(avg_time_mean, 2), round(served_mean, 2)),
  SD = c(round(avg_time_sd, 2), round(served_sd, 2))
)

# Print tables
wait_table
util_table
summary_table
```

```r
# Calculate means and standard deviations from simulation outputs
wait_mean <- colMeans(wait_mat)
wait_sd <- apply(wait_mat, 2, sd)

util_mean <- colMeans(util_mat)
util_sd <- apply(util_mat, 2, sd)

# Bar plot: Average Waiting Time per Counter (with error bars)
bar_centers <- barplot(
  wait_mean,
  names.arg = paste('Counter', 1:length(wait_mean)),
  col = 'skyblue',
  ylim = c(0, max(wait_mean + wait_sd) * 1.1),
  ylab = 'Average Waiting Time (min)',
  main = 'Average Waiting Time per Counter'
)
arrows(
  x0 = bar_centers,
  y0 = wait_mean - wait_sd,
  x1 = bar_centers,
  y1 = wait_mean + wait_sd,
  angle = 90, code = 3, length = 0.05
)

# Bar plot: Cashier Utilization per Counter (with error bars)
bar_centers2 <- barplot(
  util_mean,
  names.arg = paste('Counter', 1:length(util_mean)),
  col = 'salmon',
  ylim = c(0, max(util_mean + util_sd) * 1.1),
  ylab = 'Cashier Utilization',
  main = 'Cashier Utilization per Counter'
)
arrows(
  x0 = bar_centers2,
  y0 = util_mean - util_sd,
  x1 = bar_centers2,
  y1 = util_mean + util_sd,
  angle = 90, code = 3, length = 0.05
)



# Histogram: Average Total Time in Supermarket
hist(avg_total_time,
     breaks = 10,
     main = "Average Total Time in Supermarket",
     xlab = "Minutes",
     col = "lightblue",
     border = "black")

# Histogram: Total Customers Served Per Day
hist(n_served,
```

```
    breaks = 10,
    main = "Total Customers Served Per Day",
    xlab = "Customers",
    col = "lightgreen",
    border = "black")
```

10.2 Parameter Settings

Setting	Value
Simulation Runs	30
Opening Hours	8:00 AM – 9:00 PM (780 minutes)
Arrival Rate	$\lambda = 5$ customers/minute
Shopping Time	Log-normal(meanlog=0, sdlog=0.5)
Service Rates	1/2.5, 1/3, 1/3.5, 1/4 customers/min (Counters 1–4)

10.3 Output Summaries

	Counter	Mean_Waiting_Time	SD_Waiting_Time
Counter_1	Counter 1	907.44	42.68
Counter_2	Counter 2	1098.81	40.33
Counter_3	Counter 3	1264.63	49.76
Counter_4	Counter 4	1444.10	50.90

	Counter	Mean_Utilization	SD_Utilization
Counter_1	Counter 1	3.328	0.112
Counter_2	Counter 2	3.806	0.116
Counter_3	Counter 3	4.259	0.125
Counter_4	Counter 4	4.681	0.119

KPI	Mean	SD
Average Total Time in Supermarket	1174.28	34.97
Total Customers Served	3872.37	75.09

11 References

1. Gross, D., Shortle, J. F., Thompson, J. M., & Harris, C. M. (2013). *Fundamentals of Queueing Theory* (4th ed.). Wiley.
2. Little, J. D. C. (1961). A proof for the queueing formula: L=λW. *Operations Research*, 9(3), 383–387.
3. Queueing Theory (Wikipedia)
4. How Supermarkets Cut Checkout Queues (BBC)