# Simulation-Based Optimization

Simulation-based optimization combines simulation models with optimization techniques to find the best solutions to complex problems. It is particularly useful when dealing with systems that are too complex for analytical solutions.

## Discrete-Event Simulation (DES)

**Discrete-Event Simulation (DES) is a powerful technique used to model the operation of a supply chain as a sequence of discrete events over time. Each event occurs at a specific point in time and marks a change in the system state, such as the arrival of goods, the start of a production process, or the shipment of products to customers.**

### Key Components of DES in Supply Chain

1. **Entities: Objects that move through the system (e.g., products, vehicles, orders).**
2. **Events: Points in time when something happens (e.g., order arrival, production completion).**
3. **Resources: Elements used to perform activities (e.g., machines, workers, trucks).**
4. **Queues: Waiting lines where entities wait for resources (e.g., waiting for processing, loading).**

## Monte Carlo Simulation

**Monte Carlo Simulation (MCS) is a statistical technique that uses random sampling and probabilistic modeling to estimate the behavior of a system. In supply chain management, MCS is widely used to assess and mitigate risks, forecast demand, and optimize inventory levels.**

### Key Steps in Monte Carlo Simulation

1. **Define the Problem: Clearly state the objective of the simulation.**
2. **Develop the Model: Create a mathematical or computational model of the system.**
3. **Specify Input Distributions: Identify the uncertain variables and their probability distributions.**
4. **Run Simulations: Perform a large number of simulations to generate a distribution of possible outcomes.**
5. **Analyze Results: Evaluate the results to make informed decisions.**

## Case study - 1(DES)

### Retail Store Checkout Operations

**Objective:** Optimize checkout operations by simulating customer arrivals, checkout processes, and cashier utilization.

**Scenario:** A retail store wants to minimize customer wait times at checkout and ensure efficient use of cashiers. The goal is to improve customer satisfaction and optimize the number of cashiers needed during peak hours.

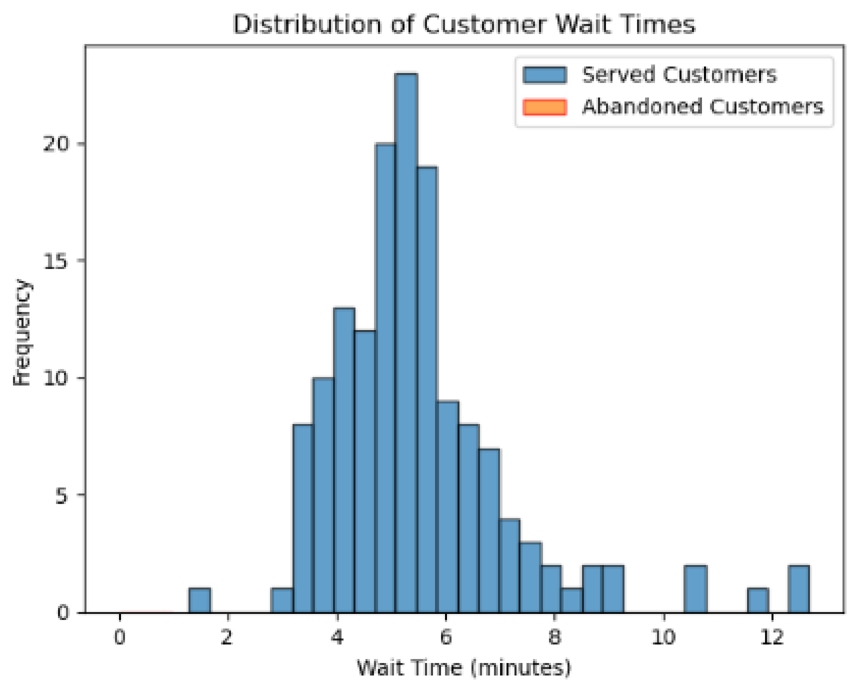
**Parameters:**

- Customer Arrival Rate: Customers arrive at the checkout counter at random intervals.
- Service Time: Time taken by a cashier to process a customer's purchase.
- Number of Cashiers: Initial number of cashiers available at the checkout counters.
- Customer Patience Time: Maximum time a customer is willing to wait in line before leaving.

**Explanation:**

1. **Define the System:** A retail store with multiple checkout counters, each managed by a cashier, where customers arrive and wait in line to be served.
2. **Model the Processes:**
   - Customer Arrival: Customers arrive at random intervals following a Poisson distribution.
   - Checkout Process: Each customer is served by a cashier for a duration that follows a normal distribution. If all cashiers are busy, customers wait in a queue.
   - Customer Patience: If a customer's wait time exceeds their patience time, they leave the queue without making a purchase.
3. **Run the Simulation:**
   - Simulate the system over a period (e.g., one business day).
   - Track metrics such as customer wait times, the number of customers served, and the number of customers who leave due to long wait times.
4. **Analyze Results:**
   - Histogram: Shows the distribution of customer wait times.
   - Average Wait Time: Provides an average of how long customers waited to be served.
   - Customer Abandonment Rate: Percentage of customers who left without being served due to long wait times.
   - Cashier Utilization: The percentage of time cashiers were busy serving customers.

```python
# Parameters
CUSTOMER_ARRIVAL_MEAN = 3  # Mean time between customer arrivals (minutes)
SERVICE_TIME_MEAN = 5  # Mean checkout time (minutes)
SERVICE_TIME_STD_DEV = 1  # Standard deviation of checkout time (minutes)
NUM_CASHIERS = 3  # Number of cashiers
CUSTOMER_PATIENCE_TIME = 10  # Max wait time before a customer leaves (minutes)
SIMULATION_TIME = 480  # Simulation time (minutes, e.g., 8 hours)

class RetailStore:
    def __init__(self, env, num_cashiers):
        self.env = env
        self.cashiers = simpy.Resource(env, num_cashiers)
        self.wait_times = []
        self.abandon_times = []

    def checkout(self, env, customer):
        arrival_time = env.now
        with self.cashiers.request() as request:
            patience_timer = env.timeout(CUSTOMER_PATIENCE_TIME)
            results = yield request | patience_timer
            if request in results:
                service_time = np.random.normal(SERVICE_TIME_MEAN, SERVICE_TIME_STD_DEV)
                yield env.timeout(service_time)
                wait_time = env.now - arrival_time
                self.wait_times.append(wait_time)
            else:
                self.abandon_times.append(env.now - arrival_time)

def customer_arrival(env, store):
    while True:
        yield env.timeout(np.random.exponential(CUSTOMER_ARRIVAL_MEAN))
        env.process(store.checkout(env, env.now))

def run_simulation(simulation_time, num_cashiers):
    env = simpy.Environment()
    store = RetailStore(env, num_cashiers)
    env.process(customer_arrival(env, store))
    env.run(until=simulation_time)
    return store.wait_times, store.abandon_times

# Run simulation
wait_times, abandon_times = run_simulation(SIMULATION_TIME, NUM_CASHIERS)

# Plot results
plt.hist(wait_times, bins=30, edgecolor='k', alpha=0.7, label='Served Customers')
plt.hist(abandon_times, bins=30, edgecolor='r', alpha=0.7, label='Abandoned Customers')
plt.title('Distribution of Customer Wait Times')
plt.xlabel('Wait Time (minutes)')
plt.ylabel('Frequency')
plt.legend()
plt.show()

# Analyze results
average_wait_time = np.mean(wait_times)
abandonment_rate = len(abandon_times) / (len(wait_times) + len(abandon_times)) * 100

print(f'Average Wait Time: {average_wait_time:.2f} minutes')
print(f'Customer Abandonment Rate: {abandonment_rate:.2f}%')
```



```
Average Wait Time: 5.51 minutes
Customer Abandonment Rate: 0.00%
```

## Case study - 2 (DES)

### Warehouse Operations

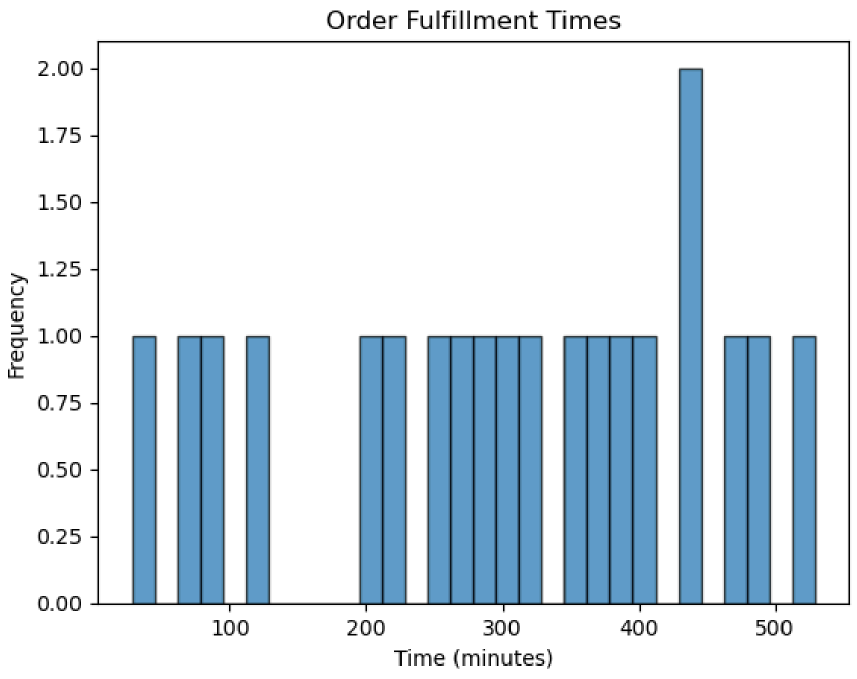
**Objective**: Optimize warehouse operations by simulating inventory flow, order processing, and resource utilization.

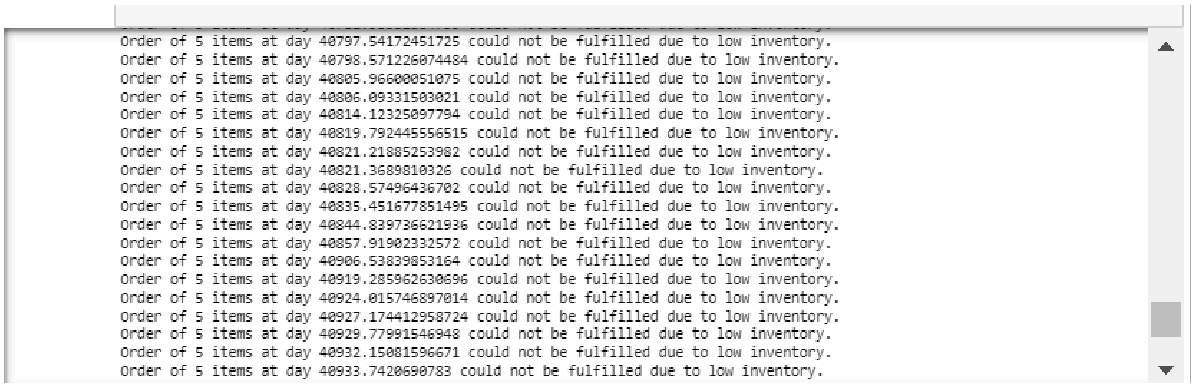
**Scenario**: A warehouse processes incoming orders and manages inventory. The goal is to minimize order fulfillment time and ensure efficient use of resources.

**Parameters**:

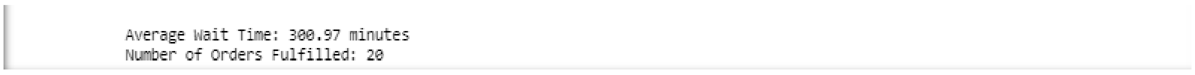
- **Order Arrival Rate**: Orders arrive at the warehouse at a random rate.
- **Processing Time**: Time taken to pick, pack, and ship an order.
- **Inventory Level**: Initial stock available in the warehouse.
- **Order Quantity**: Number of items per order.

```python
import simpy
import numpy as np
import matplotlib.pyplot as plt

# Parameters
ORDER_ARRIVAL_MEAN = 10  # Mean time between orders (minutes)
ORDER_PROCESS_TIME = 15  # Time to process an order (minutes)
INITIAL_INVENTORY = 100  # Initial inventory level
ORDER_QUANTITY = 5  # Number of items per order

class Warehouse:
    def __init__(self, env):
        self.env = env
        self.inventory = simpy.Container(env, capacity=INITIAL_INVENTORY, init=INITIAL_INVENTORY)
        self.process_order = env.process(self.order_processing())
        self.wait_times = []

    def order_processing(self):
        while True:
            # Simulate order arrival
            yield self.env.timeout(np.random.exponential(ORDER_ARRIVAL_MEAN))
            order_size = ORDER_QUANTITY
            # Check inventory
            if self.inventory.level >= order_size:
                yield self.inventory.get(order_size)
                # Simulate order processing time
                yield self.env.timeout(ORDER_PROCESS_TIME)
                self.wait_times.append(self.env.now)
            else:
                print(f'Order of {order_size} items at day {self.env.now} could not be fulfilled due to low inventory.')

def run_simulation(duration):
    env = simpy.Environment()
    warehouse = Warehouse(env)
    env.run(until=duration)
    return warehouse.wait_times

# Run simulation for 30 days
wait_times = run_simulation(30 * 24 * 60)  # 30 days in minutes

# Plot wait times
plt.hist(wait_times, bins=30, edgecolor='k', alpha=0.7)
plt.title('Order Fulfillment Times')
plt.xlabel('Time (minutes)')
plt.ylabel('Frequency')
plt.show()

print(f'Average Wait Time: {np.mean(wait_times):.2f} minutes')
print(f'Number of Orders Fulfilled: {len(wait_times)}')
```





```
Order of 5 items at day 40797.54172451725 could not be fulfilled due to low inventory.
Order of 5 items at day 40798.571226074484 could not be fulfilled due to low inventory.
Order of 5 items at day 40805.96600051075 could not be fulfilled due to low inventory.
Order of 5 items at day 40806.09331503021 could not be fulfilled due to low inventory.
Order of 5 items at day 40814.12325097794 could not be fulfilled due to low inventory.
Order of 5 items at day 40819.792445556515 could not be fulfilled due to low inventory.
Order of 5 items at day 40821.21885253982 could not be fulfilled due to low inventory.
Order of 5 items at day 40821.3689810326 could not be fulfilled due to low inventory.
Order of 5 items at day 40828.57496436702 could not be fulfilled due to low inventory.
Order of 5 items at day 40835.451677851495 could not be fulfilled due to low inventory.
Order of 5 items at day 40844.839736621936 could not be fulfilled due to low inventory.
Order of 5 items at day 40857.91902332572 could not be fulfilled due to low inventory.
Order of 5 items at day 40906.53839853164 could not be fulfilled due to low inventory.
Order of 5 items at day 40919.285962630696 could not be fulfilled due to low inventory.
Order of 5 items at day 40924.015746897014 could not be fulfilled due to low inventory.
Order of 5 items at day 40927.174412958724 could not be fulfilled due to low inventory.
Order of 5 items at day 40929.77991546948 could not be fulfilled due to low inventory.
Order of 5 items at day 40932.15081596671 could not be fulfilled due to low inventory.
Order of 5 items at day 40933.7420690783 could not be fulfilled due to low inventory.
```



```
Average Wait Time: 300.97 minutes
Number of Orders Fulfilled: 20
```

## Explanation

1. **Define the System**: A warehouse that receives and processes orders, managing inventory levels and processing times.
2. **Model the Processes**:
   - **Order Arrival**: Orders arrive at random intervals following an exponential distribution.
   - **Order Processing**: Each order is processed after a fixed duration if enough inventory is available.
3. **Run the Simulation**:
   - Simulate the system over a period (e.g., 30 days).
   - Track metrics like order fulfillment times and inventory levels.
4. **Analyze Results**:
   - **Histogram**: Shows the distribution of order fulfillment times.
   - **Average Wait Time**: Provides an average of how long orders waited to be fulfilled.

## Case study - 3 (MCS)

**Scenario:** You own a toy store and want to determine how many units of a popular toy to order for the holiday season. You need to consider uncertainties in demand and lead time.

**Step 1: Define Key Variables**

**- Demand:** The number of toys customers will buy during the holiday season.

- **Lead Time**: The time it takes for the toys to be delivered after placing an order.

**Step 2: Gather Historical Data**

- **Demand Data**: Over the past few holiday seasons, you've observed that the average demand for the toy is 500 units, with a standard deviation of 100 units.

- **Lead Time Data**: The average lead time is 2 weeks, with a standard deviation of 1 week.

**Step 3: Set Probability Distributions**

- **Demand Distribution**: Assume the demand follows a normal distribution with a mean of 500 and a standard deviation of 100.

- **Lead Time Distribution**: Assume the lead time follows a normal distribution with a mean of 2 weeks and a standard deviation of 1 week.

**Step 4: Run Simulations**

**1. Simulate Demand:** For each simulation iteration, randomly generate a demand value using the normal distribution.

**2. Simulate Lead Time:** Randomly generate a lead time value using the normal distribution.

**3. Calculate Inventory Needs:** For each iteration, determine how many toys you need to order based on the simulated demand and lead time.

**Step 5: Repeat the Simulation**

- Run the simulation for a large number of iterations (e.g., 1,000 times). Each iteration gives you a different combination of demand and lead time.

**Step 6: Analyze Results**

- After running the simulations, you will have a range of outcomes for how many toys you might need to order. For example, you might find:

  - 80% of the time, demand is between 400 and 600 units.

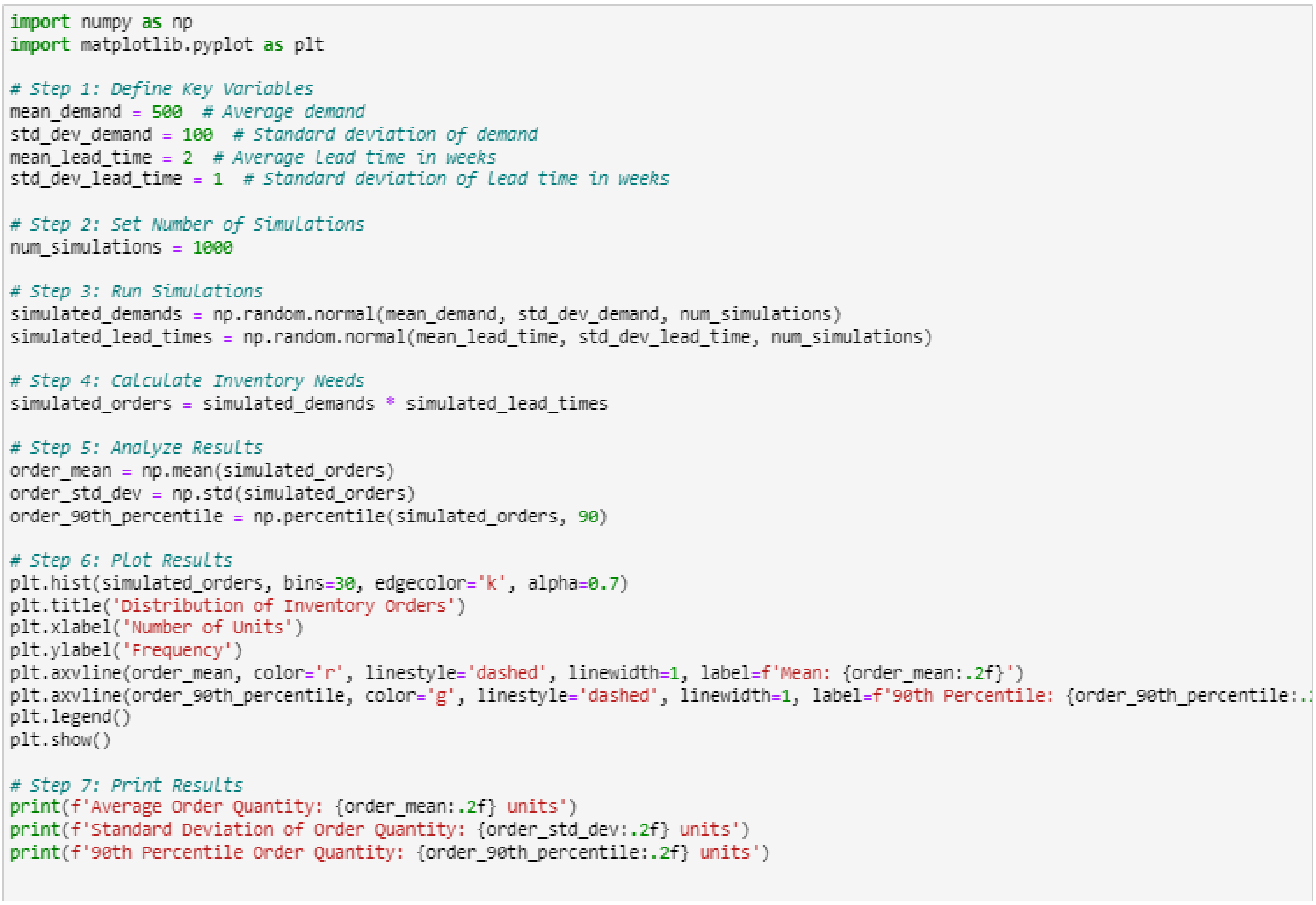
  - 10% of the time, demand exceeds 600 units.

**Step 7: Make Informed Decisions**

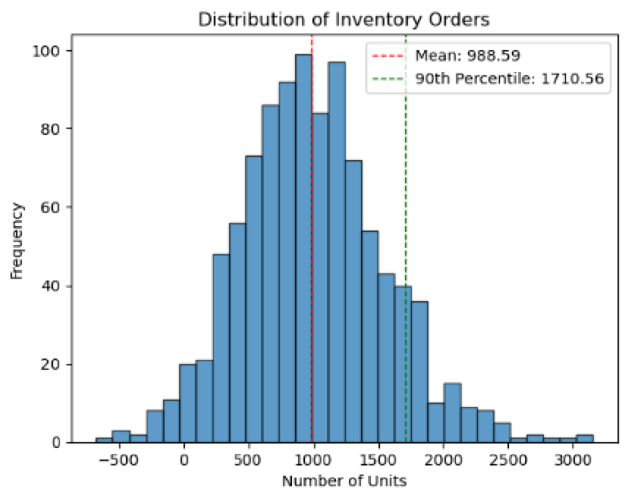
- Based on the simulation results, you can make a more informed decision about how many toys to order. If you want to ensure that you meet customer demand 90% of the time, you might decide to order 620 units, as this covers most scenarios while minimizing the risk of stockouts.

**By using Monte Carlo simulation, you can effectively account for uncertainties in demand and lead time in your supply chain management. This approach helps you make better**

**inventory decisions, ensuring that you have enough stock to meet customer needs while minimizing excess inventory.**



```python
import numpy as np
import matplotlib.pyplot as plt

# Step 1: Define Key Variables
mean_demand = 500  # Average demand
std_dev_demand = 100  # Standard deviation of demand
mean_lead_time = 2  # Average lead time in weeks
std_dev_lead_time = 1  # Standard deviation of lead time in weeks

# Step 2: Set Number of Simulations
num_simulations = 1000

# Step 3: Run Simulations
simulated_demands = np.random.normal(mean_demand, std_dev_demand, num_simulations)
simulated_lead_times = np.random.normal(mean_lead_time, std_dev_lead_time, num_simulations)

# Step 4: Calculate Inventory Needs
simulated_orders = simulated_demands * simulated_lead_times

# Step 5: Analyze Results
order_mean = np.mean(simulated_orders)
order_std_dev = np.std(simulated_orders)
order_90th_percentile = np.percentile(simulated_orders, 90)

# Step 6: Plot Results
plt.hist(simulated_orders, bins=30, edgecolor='k', alpha=0.7)
plt.title('Distribution of Inventory Orders')
plt.xlabel('Number of Units')
plt.ylabel('Frequency')
plt.axvline(order_mean, color='r', linestyle='dashed', linewidth=1, label=f'Mean: {order_mean:.2f}')
plt.axvline(order_90th_percentile, color='g', linestyle='dashed', linewidth=1, label=f'90th Percentile: {order_90th_percentile:.
plt.legend()
plt.show()

# Step 7: Print Results
print(f'Average Order Quantity: {order_mean:.2f} units')
print(f'Standard Deviation of Order Quantity: {order_std_dev:.2f} units')
print(f'90th Percentile Order Quantity: {order_90th_percentile:.2f} units')
```



```
Average Order Quantity: 988.59 units
Standard Deviation of Order Quantity: 554.44 units
90th Percentile Order Quantity: 1710.56 units
```

# Case study - 4 (MCS)

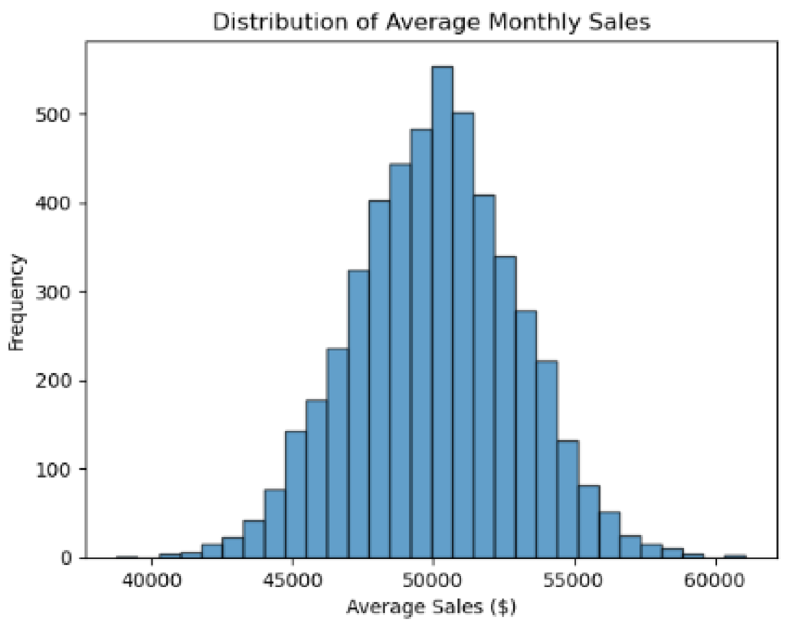
## Forecasting Monthly Sales

**Objective**: Estimate the average monthly sales for a product given historical sales data.

**Parameters**:

- Mean monthly sales: $50,000
- Standard deviation: $10,000
- Number of months: 12
- Number of simulations: 5,000

```python
import numpy as np
import matplotlib.pyplot as plt

# Parameters
mean_sales = 50000  # Mean monthly sales in $
std_dev_sales = 10000  # Standard deviation in $
num_months = 12  # Number of months in a year
num_simulations = 5000  # Number of simulations

# Run simulations
average_sales = []
for _ in range(num_simulations):
    # Generate random sample of monthly sales
    monthly_sales = np.random.normal(mean_sales, std_dev_sales, num_months)
    # Calculate the average sales for this sample
    average_sales.append(np.mean(monthly_sales))

# Convert to numpy array for analysis
average_sales = np.array(average_sales)

# Plot the distribution of average monthly sales
plt.hist(average_sales, bins=30, edgecolor='k', alpha=0.7)
plt.title('Distribution of Average Monthly Sales')
plt.xlabel('Average Sales ($)')
plt.ylabel('Frequency')
plt.show()

# Calculate the mean and confidence intervals
estimated_mean_sales = np.mean(average_sales)
confidence_interval_sales = np.percentile(average_sales, [2.5, 97.5])

print(f'Estimated Average Monthly Sales: ${estimated_mean_sales:.2f}')
print(f'95% Confidence Interval: (${confidence_interval_sales[0]:.2f}, ${confidence_interval_sales[1]:.2f})')
```



```
Estimated Average Monthly Sales: $50038.06
95% Confidence Interval: ($44367.42, $55686.25)
```