1. High-Level Experimental Design Framework

The primary objective of the computational study is to evaluate the performance of the proposed Stochastic Approximation Algorithm (SAA) for the hotel dynamic pricing problem with multiple-night stays. The experimental design will focus on assessing the solution quality, computational efficiency, scalability, and robustness of the SAA method compared to benchmark methods.

Key Components:

- **Performance Metrics:**
    - **Revenue Performance:** Total expected revenue generated.
    - **Solution Quality:** Proximity to optimal solutions where obtainable.
    - **Computational Efficiency:** CPU time and convergence rate.
    - **Scalability:** Performance across varying problem sizes.
    - **Robustness:** Performance under different demand scenarios.

- **Parameters to Vary:**
    - **Problem Size:** Number of booking periods (T) and service horizon length (N).
    - **Capacity Levels (C):** Total room capacity of the hotel.
    - **Demand Characteristics:** Arrival rates (π_t^b) and reservation price distributions $(F^b(p_t^b))$.
    - **Smoothing Parameters:** Values of α and β in the SAA method.
    - **Pricing Bounds:** Minimum and maximum allowable prices $(\underline{p}, \overline{p})$.

- **Benchmark Methods:**
    - **Deterministic Linear Programming (DLP) Approach.**
    - **Static Pricing Strategy:** Fixed prices over the booking horizon.
    - **Optimal Dynamic Programming (DP) Solution:** For small-scale

problems where computation is feasible.

- **Statistical Analysis:**
  - Use of confidence intervals to assess performance differences.
  - Hypothesis testing to determine statistical significance.
  - Regression analysis to understand the impact of parameters.

2. Detailed Breakdown of Computational Experiments

Experiment 1: Solution Quality Assessment

- **Objective:** Evaluate how closely the SAA method approximates the optimal solution.
- **Setup:**
  - **Scale:** Small problem instances (e.g., $T = 10, N = 5)$ where the optimal DP solution is computable.
  - **Parameters:** Vary capacity levels (C) and demand intensities.
- **Metrics:** Compare total revenue and solution quality between SAA and DP.
- **Statistical Analysis:** Paired t-tests to assess differences.

Experiment 2: Computational Efficiency and Scalability

- **Objective:** Assess the computational efficiency and scalability of the SAA method.
- **Setup:**
  - **Scale:** Increase T and N incrementally (e.g., T from 10 to 100, N from 5 to 50).
  - **Parameters:** Keep other factors constant to isolate the effect of problem size.
- **Metrics:** Measure CPU time, convergence rate, and memory usage.
- **Visualization:** Plot CPU time vs. problem size.

Experiment 3: Robustness Under Different Demand Scenarios

- **Objective:** Test the robustness of the SAA method under varying demand conditions.
- **Setup:**
    - **Scenarios:**
        - **High Demand:** Increase arrival probabilities (π_t^b).
        - **Low Demand:** Decrease arrival probabilities.
        - **Peak Periods:** Introduce time-dependent demand spikes.
        - **Random Fluctuations:** Add stochastic variability to demand.
    - **Parameters:** Adjust reservation price distributions accordingly.
- **Metrics:** Total revenue, booking rates, and price adjustments.
- **Statistical Analysis:** ANOVA to assess performance across scenarios.

Experiment 4: Impact of Smoothing Parameters

- **Objective:** Analyze how the smoothing parameters (α, β) affect the SAA performance.
- **Setup:**
    - **Parameters:** Vary α and β over a range of values.
    - **Metrics:** Solution quality, convergence rate, and stability.
- **Visualization:** Surface plots showing performance metrics vs. α and β.

Experiment 5: Comparison with Benchmark Methods

- **Objective:** Compare the SAA method with DLP and static pricing strategies.
- **Setup:**
    - **Parameters:** Use medium-scale problem instances (e.g., $T = 50, N = 20$).
- **Metrics:** Total revenue, computational time, and solution quality.
- **Statistical Analysis:** Compare means using t-tests; evaluate effect sizes.

3. Guidelines for Implementing Experiments in Python

Data Generation and Parameter Settings:

- **Booking Horizon (T):** Choose values such as 10, 50, 100.
- **Service Horizon (N):** Select values like 5, 20, 50.
- **Capacity (C):** Set C proportional to N (e.g., $C = 0.8 * N$).
- **Arrival Probabilities (π_t^b):** Generate using uniform or time-dependent distributions.
- **Reservation Price Distributions $(F^b(p_t^b))$:** Use a linear form for survival function $\bar{F}^b(p_t^b) = 1 - \epsilon^b p_t^b$.
- **Price Bounds $(\underline{p}, \overline{p})$:** Set based on industry standards or a range around average reservation prices.

Sample Sizes and Replications:

- **Replications:** Perform at least 30 replications per experimental condition to ensure statistical validity.
- **Random Seeds:** Use different seeds for random number generators to ensure variability.

Implementation Tips:

- **Modular Code Structure:** Separate code into modules for data generation, algorithm implementation, and analysis.
- **Efficient Data Structures:** Use NumPy arrays and pandas DataFrames for efficient computations.
- **Parallel Processing:** Utilize Python's multiprocessing library to run simulations in parallel.
- **Logging and Checkpoints:** Implement logging to track progress and save intermediate results.

4. Suggestions for Presenting and Analyzing Results

Visualization Approaches:

- **Line Charts:** Show revenue performance and computational time across different problem sizes.
- **Bar Charts:** Compare total revenue between SAA and benchmark methods.
- **Heatmaps:** Visualize the impact of smoothing parameters on performance metrics.
- **Box Plots:** Display the distribution of revenues under different demand scenarios.

Statistical Analysis:

- **Confidence Intervals:** Calculate 95% confidence intervals for key metrics.
- **Hypothesis Testing:** Use t-tests or ANOVA to determine statistical significance.
- **Regression Analysis:** Model the relationship between parameters (e.g., capacity, demand intensity) and performance.

Reporting:

- **Executive Summary:** Provide a concise overview of findings.
- **Detailed Results:** Present tables and figures with captions explaining key insights.
- **Discussion:** Interpret results, highlighting trends, anomalies, and implications.
- **Appendices:** Include supplementary material such as detailed data tables and code snippets.

5. Potential Challenges and Mitigation Strategies

Challenge 1: Computational Complexity for Large-Scale Problems

- **Mitigation:**
    - Implement algorithm optimizations (e.g., vectorization).

    - Use high-performance computing resources if available.
    - Limit the scope of large-scale experiments to manageable sizes.

Challenge 2: Convergence Issues with the SAA Method

- **Mitigation:**
    - Tune smoothing parameters (α, β) carefully.
    - Implement adaptive learning rates in the optimization algorithm.
    - Monitor convergence metrics and set appropriate stopping criteria.

Challenge 3: Data Generation Realism

- **Mitigation:**
    - Base parameter settings on industry data or realistic assumptions.
    - Validate generated data by checking statistical properties (e.g., mean, variance).
    - Conduct sensitivity analysis to assess the impact of data assumptions.

Challenge 4: Statistical Validity

- **Mitigation:**
    - Ensure sufficient sample sizes and replications.
    - Use appropriate statistical tests considering data distributions.
    - Adjust for multiple comparisons where necessary.

Additional Notes:

- **Scalability Testing:** To assess scalability, incrementally increase T and N, and observe how performance metrics change. This helps identify the limits of the SAA method.

- **Optimal Solution Comparison:** For small instances where the DP solution is computable, quantify the gap between the SAA and the optimal solution.

- **Robustness Checks:** Introduce variability not only in demand but also in other parameters like cancellation rates or no-show probabilities, if applicable.

- **Python Implementation Guidelines:**
    - **Algorithm Implementation:** Follow the provided algorithms closely, ensuring correct implementation of recursive computations and gradient updates.
    - **Code Testing:** Validate each component with unit tests before running full-scale simulations.
    - **Documentation:** Comment code thoroughly and maintain documentation for reproducibility.

Our goal is to be able to comprehensively evaluate the performance of the SAA method, providing insights into its effectiveness, efficiency, and practical applicability in dynamic hotel pricing scenarios.