1. Data Creation/Collection

For this case study, we will use a small-scale experiment to demonstrate the **Observer Pattern** and its benefits. The dataset represents a **stock price system** where multiple users (observers) need to track the price of a stock. In a real-world scenario, the stock price might change over time, and the observers (users or systems) need to be notified automatically of those changes.

The **Subject (Stock)** in this case will be a stock with a price attribute, and the **Observers** will represent users or systems that care about the stock price. For simplicity, we will start with just a small set of data.

1.1. Data Model

- **Subject**: A stock with an initial price.
- **Observers**: A small group of users or systems that react to changes in the stock price.

---

2. Experimental Procedure

The goal of this experiment is to:

- **Illustrate the methodology** of the **Observer Pattern** by demonstrating how it works in a small-scale data setup.
- **Collect empirical evidence** to show the efficiency and effectiveness of the Observer Pattern in a simple, well-defined scenario.

We will create a stock system where multiple observers will be registered to monitor changes in the stock price. When the stock price changes, each observer will be notified and will take appropriate action (for example, printing a message).

2.1. System Setup

2.1.1. Without Observer Pattern

In this scenario, the **Subject (Stock)** will **manually notify** each observer about a state change. Each observer will need to **poll** the stock to check whether there is any change, leading to a more tightly coupled system.

2.1.2. With Observer Pattern

In this scenario, the **Subject (Stock)** will **automatically notify** all registered observers about the state change, decoupling the subject and observers and ensuring that any change to the stock price is propagated to all observers seamlessly.

---

3. Code Implementation

*Found in the Code examples directory.

4. Experiment Execution

The **experiment** will involve:

1. **Stock Price Updates**: We will simulate price changes for a stock, and in both approaches, the observers will be notified.
2. **Observer Actions**: Observers will perform a simple action when notified (e.g., printing a message).
3. **Tracking Metrics**:
    - **Execution Time**: Measure the time it takes to notify all observers in both approaches.
    - **Code Complexity**: Measure how many lines of code are added/modified when new observers are added.
    - **Ease of Adding Observers**: Measure how easy it is to add/remove observers in both approaches.

---

5. Empirical Data Collection

We will collect the following **empirical data** during the experiment:

- **Code Complexity**: For each approach, we will count the number of lines of code required to add/remove observers.
    - **Without Observer Pattern**: We expect higher code complexity as we have to manually handle the interactions between observers and the subject.
    - **With Observer Pattern**: The Observer Pattern should result in less code modification when adding/removing observers.
- **Execution Time**: The time it takes to update the observers after a price change.
    - Both approaches are expected to have a time complexity of **O(N)**, where **N** is the number of observers, but the **Observer Pattern** may provide a more structured and efficient mechanism for notifying observers.
- **Ease of Adding/Removing Observers**: The time and effort required to add or remove observers.
    - **Without Observer Pattern**: Modifications to both the subject and observer are necessary.
    - **With Observer Pattern**: Observers can be added or removed independently without affecting the subject.

---

6. Results and Analysis

Without Observer Pattern

- **Code Complexity**: Higher, as the subject directly manipulates the observers and maintains manual notification.
- **Execution Time**: **O(N)** for notifying all observers when the state changes.
- **Ease of Adding/Removing Observers**: Lower, as changes are required to both the observer and subject.

With Observer Pattern

- **Code Complexity**: Lower, as the subject and observers are decoupled, and only the subject needs to notify observers.
- **Execution Time**: **O(N)** for notifying observers, but the pattern ensures that the code remains clean and manageable.
- **Ease of Adding/Removing Observers**: Much easier, as observers are managed by the subject independently.

---

7. Conclusion

The **Observer Pattern** provides significant **advantages** in terms of:

1. **Maintainability**: Less code modification is needed to add or remove observers.
2. **Scalability**: The system remains manageable as the number of observers increases.
3. **Flexibility**: New types of observers can be introduced easily without affecting the core subject.

Even though both approaches share the same **time complexity** for notifying observers (**O(N)**), the **Observer Pattern** shines in scenarios where the system needs to be flexible, scalable, and maintainable.

Chapter: Case Study in Research Report

This chapter describes the methodology and potential benefits of using the **Observer Pattern** in a simple stock price monitoring system. It provides a clear example of how the pattern works in a small-scale scenario, including the code implementation and empirical data collection.

1. **Introduction**: Overview of the Observer Pattern.
2. **System Setup**: Explanation of both approaches (with and without the Observer Pattern).
3. **Experiment Execution**: Description of the experiments, metrics, and data collection.

4. **Results and Analysis**: Presentation of empirical data comparing both approaches.
5. **Conclusion**: Summarization of the benefits and advantages of using the Observer Pattern.

This case study illustrates the real-world applicability of the **Observer Pattern** in systems where multiple entities need to track the state of a subject dynamically, making it a suitable solution for scalable and maintainable software design.