# Lernnachweis zu Kompetenz C1F

## Anwendung von Refactoring-Techniken

Die Fähigkeit, Refactoring-Techniken anzuwenden, um Code lesbarer und verständlicher zu gestalten, ist entscheidend für die Softwareentwicklung. Hier demonstriere ich die Anwendung einiger grundlegender Refactoring-Techniken an einem Beispiel:

**Ausgangscode:**

```
def calculate_mean_and_standard_deviation(data):
    n = len(data)
    total = sum(data)
    mean = total / n

    sum_squared_diff = 0
    for value in data:
        sum_squared_diff += (value - mean) ** 2

    variance = sum_squared_diff / n
    std_deviation = variance ** 0.5

    return mean, std_deviation
```

**Nach Refactoring:**

```
def calculate_mean(data):
    return sum(data) / len(data)

def calculate_variance(data, mean):
    return sum((value - mean) ** 2 for value in data) / len(data)

def calculate_standard_deviation(variance):
    return variance ** 0.5

def calculate_mean_and_standard_deviation(data):
    mean = calculate_mean(data)
    variance = calculate_variance(data, mean)
    std_deviation = calculate_standard_deviation(variance)
    return mean, std_deviation
```

**Erläuterung:**

1. Extraktion von Funktionen:
   - Die ursprüngliche Funktion wurde in kleinere Funktionen («calculate_mean», «calculate_variance», «calculate_standard_deviation») aufgeteilt, um klare und spezifische Aufgaben zu erfüllen.

2. Aussagekräftige Benennung:
   - Funktionen und Variablen wurden besser benannt, um ihre Zwecke deutlich zu machen.

3. Entfernung von Redundanz:
   - Wiederholende Berechnungen wurden vermieden, indem die Funktionen effizienter strukturiert wurden.

Diese Refactoring-Techniken verbessern die Lesbarkeit, Verständlichkeit und Wartbarkeit des Codes erheblich. Jede Funktion erfüllt nun eine klare Aufgabe, was die Gesamtstruktur transparenter macht. Die Anwendung solcher Techniken trägt dazu bei, dass der Code effizienter gepflegt und erweitert werden kann.