



Hochschule für Technik  
und Wirtschaft Berlin

University of Applied Sciences

---

# Eignung von Large Language Models (LLMs) zur Generierung von Python Code zur Datenanalyse

---

Bachelorarbeit

Name des Studiengangs  
Wirtschaftsinformatik

**Fachbereich 4**

vorgelegt von  
Maurice Krüger

Datum:  
Berlin, 15.02.2025

Erstgutachter: Prof. Dr.-Ing. Ingo Claßen

Zweitgutachter: Prof. Dr. Axel Hochstein

## Abstract

Diese Bachelorarbeit untersucht die Eignung moderner Large Language Models (LLMs) für die automatisierte Generierung von Python-Code im Kontext typischer Datenanalyseaufgaben. Dabei wird zunächst ein Überblick über die theoretischen Grundlagen von LLMs und der Programmiersprache Python gegeben. Anschließend werden in einer empirischen Untersuchung Codebeispiele durch den Marktführer ChatGPT mit dem Sprachmodell GPTo1 erzeugt und mit manuell geschriebenen Skripten verglichen. Die Bewertung erfolgt anhand mehrerer Kriterien wie Korrektheit, Performanz sowie Verständlichkeit des generierten Codes. Abschließend werden Empfehlungen für den praktischen Einsatz von LLMs in der Datenanalyse abgeleitet sowie ein Ausblick auf zukünftige Entwicklungen gegeben.

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>3</b>
1.1	Problemstellung und Forschungsfragen . . . . .	3
1.2	Relevanz der Thematik . . . . .	4
1.3	Zielsetzung . . . . .	4
1.4	Aufbau der Arbeit . . . . .	4
<b>2</b>	<b>Grundlagen</b>	<b>4</b>
2.1	Einführung Large Language Models . . . . .	5
2.2	Einführung Python . . . . .	6
2.2.1	Bedeutung und Bibliotheken . . . . .	6
2.2.2	Typische Schritte einer Datenanalyse . . . . .	6
2.3	Einführung automatisierte Code-Generierung . . . . .	6
2.3.1	Funktionsweise und Vorteile . . . . .	6
2.3.2	Herausforderungen und Grenzen . . . . .	7
2.4	Prompting mit Sprachmodellen . . . . .	7
<b>3</b>	<b>LLMs in der Programmierung – aktueller Stand</b>	<b>8</b>
3.1	Überblick und Vergleich von verschiedenen LLMs . . . . .	8
3.2	Einsatzgebiete von LLMs in der Programmierung . . . . .	10
3.3	Vergangene Studien und Arbeiten zur Code-Generierung . . . . .	10
<b>4</b>	<b>Ausgangsdaten und Testfallspezifikation</b>	<b>13</b>
4.1	Überblick und Reduktion der Datengrundlage . . . . .	13
4.2	Umwandlung in Pandas DataFrames . . . . .	14
4.3	Testfälle und Vorgehen . . . . .	14
4.3.1	Testfall 1: Sortierung und Ausgabe der Fallzahlen 2023 . . . . .	14
4.3.2	Testfall 2: Join aller Tabellen und „Bezirks-Topwert“ . . . . .	15
4.3.3	Testfall 3: Prozentuale Verteilung der Straftaten . . . . .	15
4.3.4	Testfall 4: Zeitreihe über die Jahre 2014–2023 . . . . .	16
4.3.5	Testfall 5: Ermittlung des Unterbezirks mit den meisten Raubdelikten pro Oberbezirk . . . . .	16
4.3.6	Testfall 6: Entwicklung der gefährlichsten Unterbezirke über die Jahre	17
<b>5</b>	<b>Methodik</b>	<b>17</b>
5.1	Vorgehensweise der Untersuchung . . . . .	17
5.2	Testfälle der Datenanalyse . . . . .	18
5.2.1	Testfall 1 . . . . .	18
5.2.2	Testfall 2 Verbund und Aggregation . . . . .	18
5.2.3	Testfall 3 . . . . .	19
5.2.4	Testfall 4 . . . . .	20
5.2.5	Testfall 5 . . . . .	21
5.2.6	Testfall 6 . . . . .	21

5.3	Auswertungskriterien . . . . .	23
5.4	Verwendete Tools und Daten . . . . .	23
<b>6</b>	<b>Auswertung der Python-Code-Generierung zur Datenanalyse durch LLMs</b>	<b>23</b>
6.1	Testfall 1: Sortierung und Ausgabe der Fallzahlen 2023 . . . . .	23
6.2	Testfall 2: Join aller Tabellen und Bezirks-Topwert . . . . .	25
6.3	Testfall 3: Prozentuale Verteilung der Straftaten . . . . .	27
6.4	Testfall 4: Zeitreihe über die Jahre 2014–2023 . . . . .	28
6.5	Testfall 5: Ermittlung des Unterbezirks mit den meisten Raubdelikten pro Oberbezirk . . . . .	29
6.6	Testfall 6: Entwicklung der gefährlichsten Unterbezirke über die Jahre . . . .	30
6.7	Übersicht der Ergebnisse . . . . .	32
6.8	Vergleich manuell erstellter Code . . . . .	32
<b>7</b>	<b>Fazit und Ausblick</b>	<b>33</b>
<b>8</b>	<b>Anhang</b>	<b>36</b>
8.1	Literaturverzeichnis . . . . .	36
8.2	Quellcodeverzeichnis . . . . .	37
8.2.1	Testfall 1 . . . . .	37
8.2.2	Testfall 2 . . . . .	54
8.2.3	Testfall 3 . . . . .	73
8.2.4	Testfall 4 . . . . .	91
8.2.5	Manuelle Skripte . . . . .	113
8.2.6	Komplette Antworten der AI Requests . . . . .	115

# 1 Einleitung

## 1.1 Problemstellung und Forschungsfragen

Die schnelle Entwicklung von Large Language Models (LLMs), wie zum Beispiel ChatGPT, hat in den letzten Jahren sowohl im privaten als auch im beruflichen Bereich viel Aufmerksamkeit erregt. Ursprünglich wurden LLMs hauptsächlich zur Lösung alltäglicher Probleme und der Verarbeitung und Erzeugung menschlicher Sprache eingesetzt, doch zunehmend zeigt sich, dass sie auch Programmiercode in verschiedenen Sprachen erstellen können. Besonders in der Programmiersprache Python – einer weit verbreiteten Sprache für Datenanalyse und Machine Learning – sind die Fortschritte in der automatisierten Code-Generierung durch LLMs bereits bemerkenswert[1, 2].

Aktuelle Forschungsarbeiten konzentrieren sich auf die systematische Bewertung von solch generierten Codes, um Fehlerquellen und Qualitätsmerkmale zu bemessen. Die Bereitstellung öffentlicher Evaluierungsdatensätze und -frameworks, wie etwa *HumanEval*[2] oder *Eval-Plus*[3], ermöglicht standardisierte Vergleichsstudien verschiedener LLMs. Dies eröffnet neue Anwendungsfelder im Bereich der Datenanalyse: Anstatt den Code manuell zu schreiben, könnten Nutzer in Zukunft lediglich ihre Anforderungen in natürlicher Sprache formulieren und das Modell würde diese für den Nutzer umsetzen.

Vor diesem Hintergrund stellt sich die Frage, ob und inwiefern LLMs tatsächlich qualitativ hochwertigen Python-Code für datenanalytische Aufgaben erzeugen können und wie dieser Code im Vergleich zu manuell geschriebenen Code abschneidet. Auch die möglichen Grenzen dieser automatisierten Generierung, wie etwa in Bezug auf Performanz, Wartbarkeit oder Fehlerraten, sind hierbei von großer Bedeutung.

Daraus ergibt sich die zentrale **Hauptforschungsfrage**:

*Inwieweit eignen sich Large Language Models (LLMs) zur Generierung von Python Code zur Durchführung gängiger Datenanalyseaufgaben, und wie schneidet dieser Code hinsichtlich Effizienz, Korrektheit und Wartbarkeit ab?*

Zur weiteren Strukturierung dieser Hauptfrage werden mehrere Unterfragen hinzugezogen:

- **Qualität & Korrektheit:** Wie qualitativ hochwertig ist dieser generierte Code hinsichtlich Syntax und Implementierung von Analyseaufgaben (z. B. Datenfilterung, Aggregation)?
- **Effizienz & Performanz:** Inwieweit entspricht der automatisch erzeugte Code modernen Standards bezüglich Laufzeit und Ressourcenverbrauch?
- **Wartbarkeit & Verständlichkeit:** Wie gut lässt sich der generierte Code verstehen, dokumentieren und erweitern?
- **Einsatzgebiete & Grenzen:** Für welche spezifischen Aufgaben in der Datenanalyse ist der Einsatz von LLMs sinnvoll und wo stoßen diese an ihre Grenzen?

## 1.2 Relevanz der Thematik

Die Fähigkeit, Programmiercode mit Hilfe von LLMs zu erstellen, könnte die Entwicklungsprozesse erheblich beschleunigen und neue Nutzergruppen anziehen, die bisher wenig Erfahrung mit Programmierung hatten. Besonders in der Datenanalyse können viele Arbeitsschritte – vor allem wiederkehrende Aufgaben, wie das Erstellen von Standard-Pipelines zur Datenbeschaffung- und bereinigung – automatisiert werden. Gleichzeitig gibt es jedoch Herausforderungen in Bezug auf Performanz, Wartbarkeit und Transparenz.

## 1.3 Zielsetzung

Das Ziel dieser Arbeit ist es, herauszufinden, wie gut moderne LLMs (Large Language Models) für die automatische Code-Generierung in der Datenanalyse mit Python geeignet sind. Dafür wird in einem Experiment Code von einem LLM generiert und mit manuell geschriebenem Code verglichen. Der Vergleich basiert auf Kriterien wie Korrektheit, Performance und Wartbarkeit. Auf Basis der Ergebnisse werden Empfehlungen für den Einsatz von LLMs in der Praxis gegeben und deren Grenzen diskutiert. Zum Schluss wird ein Ausblick darauf gegeben, wie sich diese Technologie in Zukunft weiterentwickeln könnte und welche Auswirkungen das auf Aufgaben in der Datenanalyse haben könnte.

## 1.4 Aufbau der Arbeit

Nach dieser Einleitung (Kapitel 1) folgt in Kapitel 2 eine Darstellung der **Grundlagen**, dazu gehört eine Einführung in Large Language Models, die Programmiersprache Python, die automatisierte Code-Generierung und Prompting. Kapitel 3 gibt einen Überblick über den aktuellen Stand der Forschung, in dem verschiedene LLM-Modelle, Publikationen und Evaluationstechniken vorgestellt werden. Darauf aufbauend werden in Kapitel 4 die **Ausgangsdaten** und eine genaue **Testfallspezifikation** und in Kapitel 5 die **Methodik** der Arbeit erläutert. Kapitel 6 enthält dann die **Auswertung** der gewonnenen Daten sowie den Vergleich von durch ein LLM generierten und manuell geschriebenen Code. Kapitel 7 fasst die Ergebnisse zusammen, beantwortet die Forschungsfragen und gibt einen **Ausblick** auf weitere Entwicklungen. Schließlich enthält Kapitel 8 den **Anhang**, einschließlich Literaturverzeichnis und relevanter Dokumentationen.

# 2 Grundlagen

Im folgenden Kapitel werden die theoretischen und technischen Grundlagen vorgestellt, die für das Verständnis dieser Arbeit notwendig sind. Abschnitt 2.1 beschäftigt sich mit den Large Language Models, ihrer Funktionsweise und ihrer Bedeutung in der Code-Generierung. Danach wird in Abschnitt 2.2 das Potenzial der Programmiersprache Python für die Datenanalyse erläutert, bevor Abschnitt 2.3 das Konzept der automatisierten Code-Generierung behandelt und Abschnitt 2.4 eine kurze Einführung in Prompting gibt.

## 2.1 Einführung Large Language Models

Bei Large Language Models (LLMs) handelt es sich um eine spezielle Klasse von vortrainierten Sprachmodellen (so genannten *Pre-trained Language Models* (PLMs)) mit einer besonders großen Anzahl an Parametern – typischerweise im Bereich von mehreren zehn bis hundert Milliarden. Sie entstanden als Weiterentwicklung früherer Sprachmodellierungsansätze, die sich über viele Jahre hinweg von *Statistical Language Models* (SLMs) hin zu *Neural Language Models* (NLMs) entwickelt haben. Ein wesentlicher Meilenstein in dieser Entwicklung war die Einführung von Transformer-basierten PLMs, die auf großen Textkorpora trainiert wurden und herausragende Leistungen bei verschiedenen Aufgaben der natürlichen Sprachverarbeitung (*Natural Language Processing* (NLP)) erzielten. In den Forschungen entdeckte man, dass eine Vergrößerung der Modellgröße nicht nur zu erheblichen Leistungssteigerungen führt, sondern dass LLMs ab einer bestimmten Größenordnung auch neue Fähigkeiten entwickeln – beispielsweise *In-Context Learning*, das es ihnen ermöglicht, Aufgaben ohne explizites Nachtrainieren zu lösen. Um diese leistungsfähigeren Modelle von kleineren abzugrenzen, hat sich der Begriff Large Language Models (LLMs) etabliert[4].

Die schon genannten Transformer-Architekturen, welche einen Self-Attention-Mechanismus nutzen, ermöglichen es dem Modell Beziehungen zwischen verschiedenen Wörtern oder Tokens in einer Eingabe zu erkennen, wobei es nicht von Bedeutung ist, an welcher Position diese stehen. Die Transformer-Architektur unterscheidet sich desweiteren von früheren Architekturen, wie zum Beispiel *Recurrent Neuronal Networks* (RNNs) dadurch, dass sie auf die rekursive Verarbeitung der Tokens verzichtet und stattdessen alle Tokens parallel verarbeitet, wodurch die LLMs deutlich effizienter auf Eingaben reagieren können. Diese Leistung der LLMs korreliert jedoch stark mit der Größe der Modelle und der Menge an Trainingsdaten, was dazu führt, dass besonders gute Modelle einen deutlich höheren Ressourcenbedarf und eine deutlich größere Menge an Trainingsdaten benötigen[5].

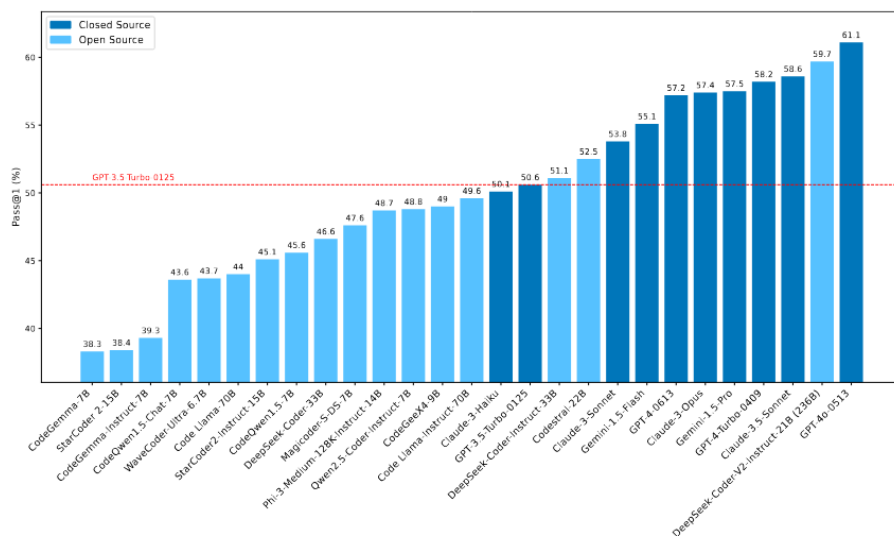


Abbildung 1: Leistungsvergleich verschiedener Modellgrößen (nach Jiang et al. (2024), basierend auf 'A Survey on Large Language Models for Code Generation').

In den letzten Jahren wurden mehrere Benchmarks und Evaluierungsdatensätze speziell für die Code-Generierung entwickelt. Beispiele dafür sind *HumanEval*[2] und *EvalPlus*[3], die genutzt werden, um die Genauigkeit und Zuverlässigkeit von LLMs in verschiedenen Programmiersprachen zu testen. Erste Studien zeigen, dass LLMs einfache bis mittelschwere Aufgaben oft vollständig lösen können. Bei komplexeren oder sehr speziellen Aufgabenbereichen stoßen sie aber noch an ihre Grenzen[1]. Obwohl LLMs in vielen Sprachen Code generieren können, hat sich Python als einer der Hauptfoki herauskristallisiert. Dies liegt an der weit verbreiteten Nutzung von Python in Wissenschaft und Industrie, insbesondere in den Bereichen Datenanalyse und Machine Learning.

## 2.2 Einführung Python

### 2.2.1 Bedeutung und Bibliotheken

Python ist dank seiner Syntax, aktiven Community und vielen hilfreichen Bibliotheken eine der am weitesten verbreiteten Sprachen für Datenanalyse. Ein paar der wichtigsten Bibliotheken, die in der Datenanalyse verwendet werden, sind:

- **pandas** – Datenstrukturen und -bearbeitung,
- **NumPy** – numerische Berechnungen,
- **scikit-learn** – Machine-Learning-Algorithmen,
- **Matplotlib** – Visualisierung,

Diese und weitere Bibliotheken ermöglichen eine effiziente Umsetzung datenanalytischer Projekte und werden bereits in LLM-Trainings berücksichtigt, wodurch generierter Code auf bekannte Funktionen zurückgreifen kann[3, 2].

### 2.2.2 Typische Schritte einer Datenanalyse

Die Grundschrirte einer klassischen Datenanalyse in Python enthält folgende Schritte:

1. *Datenimport* (z. B. CSV-Dateien, Datenbanken, APIs),
2. *Datenbereinigung* (fehlende Werte, Duplikate, Datentypen),
3. *Analyse und Visualisierung* (Statistiken, Plots),

Im Rahmen dieser Arbeit wird untersucht, ob LLMs diese Schritte automatisieren können und an welchen Stellen manuell eingegriffen werden muss.

## 2.3 Einführung automatisierte Code-Generierung

### 2.3.1 Funktionsweise und Vorteile

Automatisierte Code-Generierung mithilfe von LLMs basiert auf *Prompts*, also Benutzeranfragen in natürlicher Sprache. LLMs haben hierbei die Möglichkeit sich flexibel an den vom



Benutzer gegebenen Kontext anzupassen und können die natürliche Sprache in funktionsfähigen Code umwandeln. Ebenso müssen LLMs nicht spezifisch auf eine Aufgabe trainiert werden, aufgrund der großen Trainingsdaten, die ihnen zur Verfügung stehen[2]. Insbesondere für datenanalytische Aufgaben, bei denen standardisierte Skripte (z. B. für das Einlesen und Bereinigen von Daten) immer wieder benötigt werden, kann dies zu einer erheblichen Zeiterparnis führen und ermöglicht die Nutzung von LLMs auch für weniger erfahrene Personen, die nicht über tiefgreifende Programmierkenntnisse verfügen.

### 2.3.2 Herausforderungen und Grenzen

Trotz beeindruckender Fortschritte stößt die automatisierte Code-Generierung noch häufig an Grenzen:

- **Komplexe Datenstrukturen:** LLMs zeigen teils Schwächen bei Aufgaben mit hochgradiger Komplexität oder spezifischem Wissen, wenn zu wenig Kontext durch den Nutzer gegeben wird[6].
- **Performanz:** Generierter Code ist nicht immer optimal hinsichtlich Laufzeit oder Speicherverbrauch[7].
- **Wartbarkeit:** Kommentare, klare Code-Struktur und Dokumentation fehlen häufig[6].
- **Fehleranfälligkeit:** Auch Code, der vorerst funktionsfähig erscheint, kann immer noch Bugs oder Sicherheitslücken enthalten[2, 6].

Wie präsent diese Herausforderungen in datenanalytischen Aufgaben sind, soll in den folgenden Kapiteln untersucht werden. Vor allem durch den Vergleich von generiertem und manuell geschriebenem Code lassen sich die Stärken und Schwächen von LLMs in der Datenanalyse besser einschätzen.

## 2.4 Prompting mit Sprachmodellen

Mit dem Begriff *Prompting* wird das Verfahren beschrieben, ein zuvor trainiertes Sprachmodell allein durch spezifische Eingabetexte (*Prompts*) zu steuern, ohne weiteres trainieren des Modells. Im Gegensatz zur zuvor verbreiteten Vorgehensweise, ein vortrainiertes Modell für jede Aufgabe mit allen notwendigen Parametern komplett anzupassen, wird beim Prompting direkt auf das bereits eingearbeitete Wissen des Modells zurückgegriffen und steuert dadurch dessen Ausgabe durch passende Formulierungen[8]. Mit der Veröffentlichung großer Modelle wie GPT-3 zog dieses Vorgehen große Aufmerksamkeit auf sich, weil es diesen Modellen ermöglichte, durch spezifische Anweisungen in einer Prompt, komplexe Aufgaben zu lösen, wie Brown et al.(2020)[9] zeigten.

### Grundlegende Strategien

- **Zero-Shot, One-Shot und Few-Shot Prompting:** Bei *Zero-Shot* Prompting wird dem Modell lediglich eine Aufgabenbeschreibung gegeben, ohne Beispiele. Hierbei soll das Modell von alleine auf die richtige und gewünschte Lösung kommen. Bei *One-Shot*

wird dem genau ein Beispiel hinzugefügt, während *Few-Shot* mehrere Demonstrationsbeispiele bereitstellt. Erste Arbeiten, wie jene von Brown et al.[9], zeigten, dass schon wenige Beispiele im Prompt teils große Leistungsunterschiede bewirken können.

- **Instruction-based Prompting:** Anstatt nur Beispiele zu geben, werden präzise Anweisungen in Textform formuliert, wie beispielsweise „*Fasse den Text in drei Sätzen zusammen.*“. Ouyang et al.(2022) führen dafür auch ihr Modell *InstructGPT* ein, welches speziell darauf trainiert wurde, solche Anweisungen verlässlich und mit Berücksichtigung der Wünsche des Nutzers in korrekter Weise umzusetzen[10].
- **Chain-of-Thought Prompting:** Hierbei wird das Modell dazu angewiesen Schrittweise vorzugehen und unter Umständen diese Zwischenschritte explizit auszugeben. Durch das schrittweise abarbeiten der Aufgaben liefern Modelle oft bessere und eher nachvollziehbare Ergebnisse zurück, wie Wei et al.(2022) zeigen[11].

### 3 LLMs in der Programmierung – aktueller Stand

Die Entwicklung von Large Language Models (LLMs) hat in den letzten Jahren nicht nur die Art und Weise, wie natürliche Sprache verarbeitet und generiert wird, verändert, sondern auch große Fortschritte in der automatisierten Code-Erstellung ermöglicht. Durch die Kombination aus leistungsstarken Modellarchitekturen wie Transformers, großen Mengen an Trainingsdaten und moderner Hardware haben LLMs heute eine große Präsenz in vielen Bereichen der Softwareentwicklung. In diesem Kapitel werden die aktuellen Entwicklungen und verfügbaren Modelle vorgestellt. Außerdem wird ein Überblick über ihre Einsatzmöglichkeiten in der Softwareentwicklung und Datenanalyse gegeben. Zum Schluss werden wichtige Studien und Arbeiten zur Code-Generierung betrachtet, darunter etwa die von Chen et al. (2021) vorgestellte Arbeit zu Codex, einem Modell, das speziell für die automatisierte Programmierung entwickelt wurde[2] und die von Liu et al. (2023) veröffentlichte Arbeit zur Evaluation von generiertem Code mithilfe von *EvalPlus*[1].

#### 3.1 Überblick und Vergleich von verschiedenen LLMs

Derzeit existiert eine Vielzahl an LLMs, darunter auch viele, die gezielt zur Code-Generierung entwickelt wurden. Zu den bekanntesten Beispielen zählen ChatGPT (GPTo1 als das modernste Modell), OpenAI Codex, Code Llama[12], StarCoder [13], CodeT5[14] oder CodeGen[15].

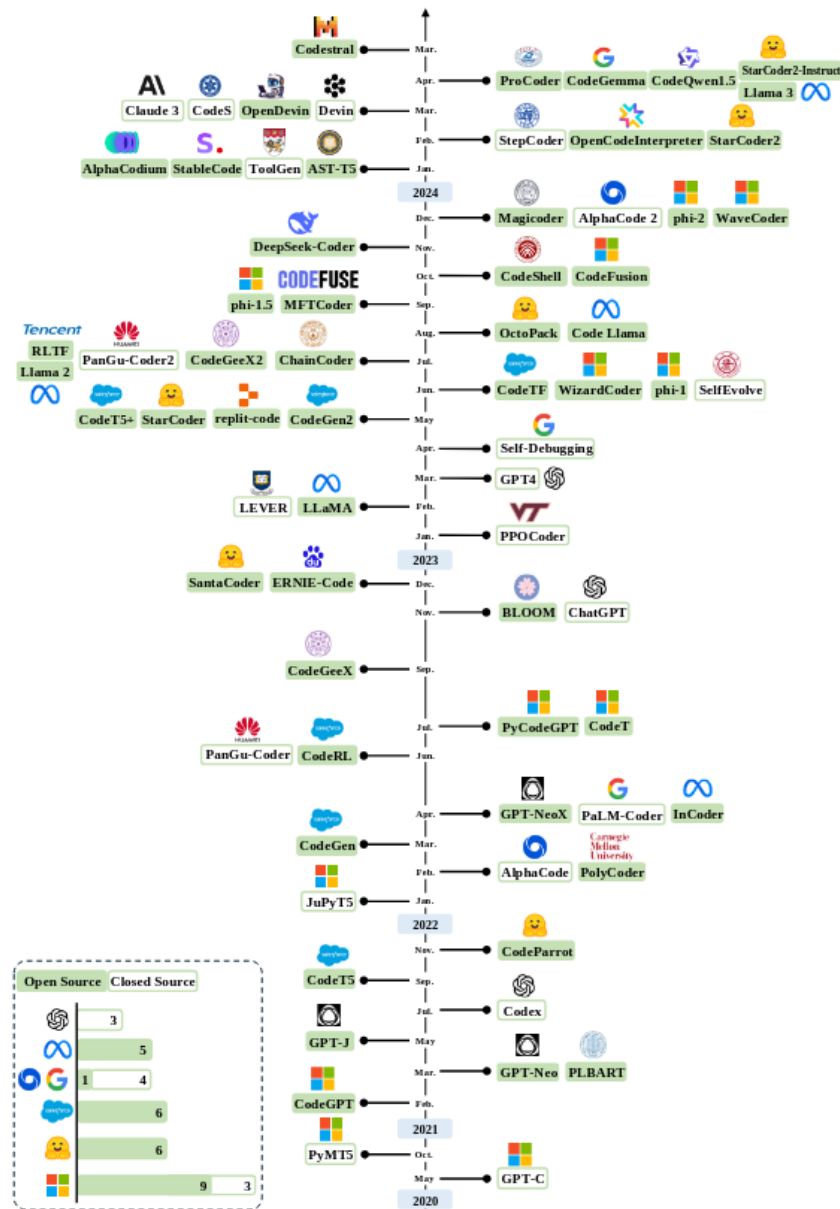


Abbildung 2: Chronologische Übersicht von Large Language Models für die Code Generierung der letzten Jahre (nach Jiang et al. (2024), basierend auf 'A Survey on Large Language Models for Code Generation').

Diese Modelle teilen sich häufig folgende Merkmale:

1. **Transformer-Architektur:** Nahezu alle modernen LLMs beruhen auf dem Transformer-Modell.
2. **Große Parameteranzahl:** Typische LLMs verfügen über eine Vielzahl an Parametern und benötigen entsprechend umfangreiche Trainingsdaten, zu denen in vielen Fällen öffentlich verfügbare Code-Repositories (z. B. GitHub) zählen[2].
3. **Breite Sprachenunterstützung:** Neben Python werden häufig Java, JavaScript und andere Programmiersprachen abgedeckt[2, 5].

Ein Vergleich der LLMs lässt sich anhand verschiedener Kriterien vornehmen:

- **Größe und Trainingsdaten:** Modelle wie GPT-4 oder Code Llama sind mit einer Vielzahl an Code-Datensätzen trainiert und erreichen dadurch in Benchmarks eine hohe Erfolgsquote[1].
- **Lizenz und Offenheit:** Neben proprietären Modellen, wie GitHub Copilot und ChatGPT, existieren mit Code Llama, StarCoder[13] oder CodeGen[15] auch Open Source Alternativen.
- **Spezialisierung:** Einige Modelle sind speziell auf Code-Generierung abgestimmt (z.B. Code Llama, StarCoder[13]), wohingegen andere (z.B. ChatGPT) einen generellen Sprachkontext haben, um auch andere Fragen zu beantworten, der sich jedoch auch auf Code-Aufgaben anwenden lässt.

### 3.2 Einsatzgebiete von LLMs in der Programmierung

Die zunehmende Leistungsfähigkeit von Large Language Models (LLMs) ermöglicht es, Programmieraufgaben in diversen Bereichen zu automatisieren oder zu beschleunigen. Häufig genannte *Einsatzgebiete* sind dabei:

- **Code-Generierung:** Ermöglicht die Code-Generierung auf Grundlage von Beschreibungen aus natürlicher Sprache[2]. Ebenso bieten manche Modelle die Möglichkeit zu fertigen Funktionen Tests zu generieren, um dessen Funktionalität zu überprüfen.
- **Autovervollständigung:** Integriert in Entwicklungsumgebungen wie Visual Studio Code können Tools wie GitHub Copilot repetitive Abläufe direkt im Code vervollständigen oder Vorschläge zur Vervollständigung von neu begonnenem Code liefern[2].
- **Refactoring und Fehlersuche:** Dank ihrer Kontextsensitivität können LLMs bestehenden Code analysieren und an einigen Stellen Möglichkeiten zur Optimierung oder Korrektur vorschlagen[2, 14]. Dadurch lassen sich Bugs, Redundanz und ineffiziente Code-Strukturen frühzeitig identifizieren und beheben.
- **Automatisierte Dokumentation und Code-Kommentierung:** Viele Modelle bieten die Möglichkeit vorhandenen Code zu analysieren und dazu Kommentarblöcke oder gar ganze Dokumentationen zu erstellen[14, 5].

Obwohl diese Einsatzgebiete großes Potenzial bieten, sind LLMs nicht frei von Fehlern. Gerade bei komplexen Entscheidungen zur Programm- und Codearchitektur können diese oft mit dem Level durch das menschliche Fachwissen nicht mithalten [16].

### 3.3 Vergangene Studien und Arbeiten zur Code-Generierung

Die Forschung zur automatisierten Code-Generierung hat in den letzten Jahren eine rasante Entwicklung erlebt, wobei Arbeiten aus den Bereichen *Software Engineering*, *Large Language Models* und *Machine Learning* zusammenfließen. Jiang et al. (2024) haben in ihrer Arbeit “A Survey on Large Language Models for Code Generation“ eine Übersicht über die Entwicklung

der veröffentlichten Arbeiten zu LLMs und Software Engineering erstellt, welche in Abbildung 3 dargestellt ist.

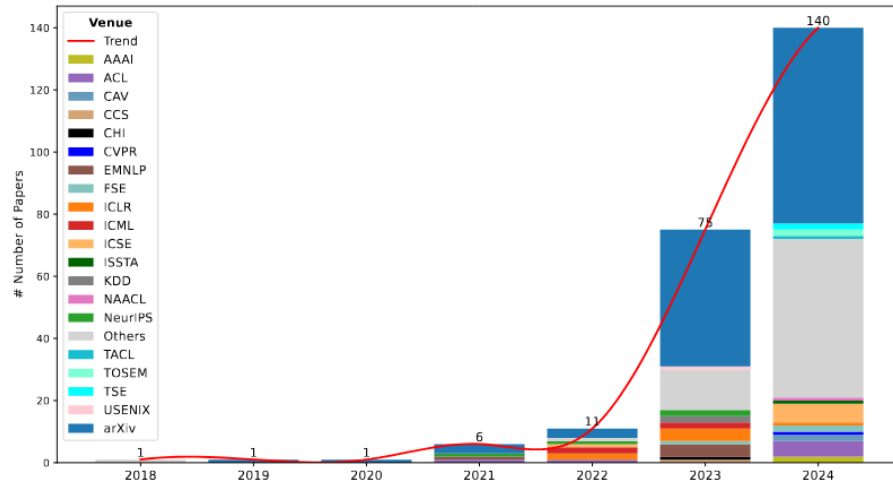


Abbildung 3: Übersicht der Verteilung von veröffentlichten Arbeiten zu LLMs und Software Engineering der letzten Jahren (nach Jiang et al. (2024), basierend auf 'A Survey on Large Language Models for Code Generation'[5]).

Im Folgenden werden einige vergangene Studien/Arbeiten zur Code Generierung mit LLMs vorgestellt:

1. **“Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation”** von Liu et al. (2023) [1]:  
In diesem Paper wird untersucht, wie korrekt der von LLMs wie ChatGPT, Code Llama etc. generierte Code ist. Dafür wird *EvalPlus* eingeführt. Dies ist ein neues Evaluierungsframework, das bestehende Testdatensätze wie *HumanEval* durch weitere automatisierte Testfälle erweitert. Hier kommen Liu et al. zu dem Ergebnis, dass bisher viele Fehler in generiertem Code nicht erkannt wurden, wodurch die Modelle in ihrer Leistung überschätzt wurden. Die Autoren weisen darauf hin, wie wichtig umfassende Tests sind, um die tatsächliche Funktionalität der LLMs für die Codegenerierung zu bewerten.
2. **“Evaluating Large Language Models Trained on Code”** von Chen et al. (2021) [2]:  
In diesem Paper wird Codex vorgestellt. Dies ist ein LLM, das speziell auf öffentlich verfügbarem Code von Github trainiert wurde, um dessen Fähigkeiten Python Code zu schreiben zu analysieren. Dies wird mithilfe des HumanEval-Datensatzes untersucht. Hierbei soll im genauen Python-Code aus Docstrings generiert und dieser dann bewertet werden. Die Ergebnisse zeigen, dass Codex im Vergleich zu anderen Modellen wie GPT-3 deutlich besser abschneidet, jedoch bei komplexeren Aufgaben seine Grenzen erreicht. Eine mehrfach wiederholte Lösungsgenerierung verbessert die Erfolgsrate, was das Potenzial ihres Ansatzes verdeutlicht.

	Size	pass@k	k=1*	k=1	k=10	k=100	T <sub>1</sub> *	T <sub>10</sub> *	T <sub>100</sub> *
GPT-4 [49]	N/A	base	88.4						
		+extra	76.2						
Phind-CodeLlama [52]	34B	base	71.3	71.6	90.5	96.2	.2	.8	.8
		+extra	67.1	67.0	85.0	92.5	.2	.8	.8
WizardCoder-CodeLlama [38]	34B	base	73.2	61.6	85.2	94.5	.2	.8	.8
		+extra	64.6	54.5	78.6	88.9	.2	.8	.8
ChatGPT [48]	N/A	base	73.2	69.4	88.6	94.0			
		+extra	63.4	62.5	82.1	91.1			
CODELLAMA [54]	34B	base	51.8	52.0	82.4	95.0	.2	.8	.8
		+extra	42.7	43.1	73.7	89.4	.2	.8	.8
	13B	base	42.7	44.6	77.6	92.7	.4	.8	.8
		+extra	36.6	37.4	69.4	88.2	.4	.8	.8
	7B	base	37.8	39.2	69.1	89.7	.2	.8	.8
		+extra	34.1	34.5	61.4	82.9	.2	.8	.8
StarCoder [13]	15B	base	34.1	32.2	56.7	84.2	.2	.8	.8
		+extra	29.3	27.8	50.3	75.4	.2	.8	.8
CodeGen [46]	16B	base	32.9	32.2	56.0	81.5	.2	.6	.8
		+extra	26.8	27.2	48.4	71.4	.2	.6	.8
	6B	base	29.3	27.7	46.9	72.7	.2	.6	.8
		+extra	25.6	23.6	41.0	64.6	.2	.6	.8
	2B	base	24.4	18.4	39.8	66.8	.2	.8	.8
		+extra	20.7	15.1	34.8	55.8	.2	.2	.8
CODET5+ [64]	16B	base	31.7	32.2	58.5	83.5	.2	.6	.8
		+extra	26.2	27.4	51.1	76.4	.2	.6	.8
MISTRAL [26]	7B	base	28.7	28.1	55.2	83.8	.2	.8	.8
		+extra	23.8	23.7	48.5	76.4	.2	.8	.8
CodeGen2 [45]	16B <sup>4</sup>	base	19.5						
		+extra	16.5						
	7B	base	18.3	17.9	30.9	50.9	.2	.6	.8
		+extra	16.5	15.9	27.1	45.4	.2	.6	.8
	3B	base	15.9	15.2	23.9	38.6	.2	.4	.8
		+extra	12.8	12.9	21.2	34.3	.2	.4	.8
	1B	base	11.0	10.2	15.1	24.7	.2	.6	.6
		+extra	9.1	8.7	13.7	21.2	.2	.6	.6
VICUNA [12]	13B	base	16.5	15.3	30.1	54.8	.2	.8	.8
		+extra	15.2	13.9	25.8	46.7	.2	.8	.8
	7B	base	11.6	10.9	23.8	42.3	.2	.6	.6
		+extra	11.0	10.3	20.3	35.0	.2	.6	.6
SantaCoder [2]	1.1B	base	14.6	16.6	29.2	45.4	.4	.6	.8
		+extra	12.8	14.2	26.2	40.6	.4	.6	.8
INCODER [18]	6.7B	base	15.9	15.6	27.7	45.0	.2	.4	.6
		+extra	12.2	12.4	22.2	38.9	.2	.6	.6
	1.3B	base	12.2	10.0	15.9	25.2	.2	.6	.6
		+extra	10.4	7.9	13.5	20.7	.2	.6	.4
GPT-J [63]	6B	base	12.2	11.3	17.7	31.8	.2	.6	.6
		+extra	10.4	9.5	15.2	25.9	.2	.6	.6
GPT-NEO [5]	2.7B	base	7.9	6.5	11.8	20.7	.2	.6	.6
		+extra	6.7	6.0	9.0	16.8	.2	.6	.6
PolyCoder [70]	2.7B	base	6.1	5.9	10.2	17.1	.2	.4	.6
		+extra	5.5	5.3	7.9	13.6	.2	.6	.6
StableLM [60]	7B	base	2.4	2.7	7.5	15.8	.2	.6	.6
		+extra	2.4	2.6	6.2	11.9	.2	.6	.6

Abbildung 4: Vergleich der Ergebnisse von *HumanEval* und *EvalPlus* (nach Liu et al. (2023), basierend auf “Is your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation”[1]).

### 3. “A Survey on Large Language Models for Code Generation” von Jiang et al. (2024) [5]:

Dieses Paper gibt einen allgemeinen und umfassenden Überblick über den aktuellen Forschungsstand zu LLMs für die Codegenerierung. Es greift Themen wie Datenaufbereitung, Modellarchitekturen und Benchmarks auf. Zudem werden Herausforderungen, wie die praktische Einführung und ethische Fragen diskutiert. Die Autoren leiten sich wichtige Forschungsfragen ab und verdeutlichen, dass LLMs in der Codegenerierung große Fortschritte gemacht haben, aber es weiterhin Potenzial zur Optimierung gibt.

4. **“Evaluating Language Models for Efficient Code Generation”** von Liu et al. (2024) [17]:

Auch in dieser Arbeit von Jiawei Liu wird die Effizienz von Code untersucht, welcher von LLMs generiert wird. Hierbei mit Fokus auf Performance und Ressourcennutzung. Dafür wird *Differential Performance Evaluation (DPE)* entwickelt und der *EvalPerf*-Benchmark eingeführt. Dieser enthält komplexere Programmieraufgaben als der zuvor eingeführt *EvalPlus*. Hier kommt man zu dem Entschluss, dass größere Modelle nicht automatisch auch effizienteren Code erzeugen. Stattdessen werden Effizienz und Korrektheit des Codes durch *Instruction Tuning*(gezieltes Trainieren des Modells, um besser auf Anweisungen in natürlicher Sprache zu reagieren) verbessert.

Zusammenfassend zeigen die genannten Studien, dass LLMs zwar großes Potenzial zur automatisierten Code Generierung besitzen, sie aber immer noch Probleme aufweisen und menschliche Entwickler nicht komplett ersetzen können. Besonders bei komplexeren Aufgaben, spezifischen Anforderungen oder Fragen zur Softwarearchitektur stoßen sie an ihre Grenzen.

## 4 Ausgangsdaten und Testfallspezifikation

### 4.1 Überblick und Reduktion der Datengrundlage

Die Datengrundlage für die empirische Untersuchung bilden Kriminalitätsstatistiken (sogenannte *Fallzahlen*) der Stadt Berlin[18]. Die zugehörige Excel-Datei umfasst mehrere Sheets, einmal die genauen Fallzahlen und einmal die Häufigkeitszahlen, jeweils zu den Jahren 2014–2023. Darin sind die Straftaten pro Bezirk (bzw. Ober- und Unterbezirke) aufgelistet. In der Ursprungsform gliedert sich die Tabelle wie folgt:

- **Oberbezirk:** Enthält aggregierte Zahlen der jeweiligen Unterbezirke.
- **Unterbezirke:** Ausführlichere Aufschlüsselung der Straftaten innerhalb des Oberbezirks.
- **Spalten mit Straftat-Kategorien:** u. a. „Straftaten insgesamt“, „Körperverletzungen“, „Diebstahl“, ...

Für die Analyse der ersten vier Testfälle wird jedoch nur auf *Oberbezirks*-Daten zurückgegriffen. Die Unterbezirke werden **nicht** berücksichtigt, um die Komplexität zu reduzieren und das Fokusgebiet auf übergeordnete Bezirke zu legen. Ziel ist eine übersichtlichere Zusammenfassung, bei der „Oberbezirk“ die wichtigste Bezugsgröße ist. Ebenso werden nicht verwendete Sheets aus der Excel-Datei und Zeilen, die nicht zur Tabelle der Daten dazugehören entfernt und die Spaltennamen vereinheitlicht.

Anschließend werden zwei weitere Testfälle durchgeführt, bei denen mit dem originalen Datensatz gearbeitet wird, um feststellen zu können, wie gut LLMs mit komplexeren Datenstrukturen umgehen können und ob es eine Korrelation zwischen der Komplexität der Daten und der Qualität des generierten Codes gibt.

## 4.2 Umwandlung in Pandas DataFrames

Jedes für den jeweiligen Testfall notwendige Jahr (bzw. jedes Sheet) soll in einer separaten `pandas`-DataFrame-Tabelle abgebildet werden. Dies geschieht folgendermaßen:

1. **Reduzieren der Datengrundlage:** Entfernen der Unterbezirke, um nur die Oberbezirke zu behalten.
2. **Bereinigung und Umbenennung:** Unnötige Spalten werden entfernt, Spaltennamen ggf. standardisiert (z. B. „Bezirk“, „Straftaten\_insgesamt“).
3. **Einlesen der Excel-Sheets:** Mit `pandas.read_excel(...)` wird jede Jahres-Tabelle eingelesen.
4. **Speicherung in DataFrame:** Pro Sheet entsteht ein bereinigtes und vereinheitlichtes DataFrame.

Hierbei ist wieder zu beachten, dass die ersten beiden Schritte lediglich für die ersten vier Testfälle durchgeführt werden. Die beiden weiteren Testfälle arbeiten mit dem originalen Datensatz.

## 4.3 Testfälle und Vorgehen

Insgesamt sind sechs Testfälle definiert, die unterschiedliche Aspekte der Datenanalyse abdecken. Für jeden Testfall werden **15 Ausführungen** erzeugt, wobei drei verschiedene Prompting-Strategien (jeweils fünf Wiederholungen) zum Einsatz kommen:

- **Strategie A: Prompt wie ein „normaler User“**  
Hier wird eine einfache, natürlichsprachliche Anfrage formuliert, ohne viele zusätzliche Informationen.
- **Strategie B: Prompt mit Metadaten**  
In diesem Ansatz werden neben der eigentlichen Anfrage auch relevante Details wie Spaltennamen oder Strukturhinweise explizit übergeben.
- **Strategie C: Prompt mit *Chain of Thought***  
Das Modell erhält schrittweise Gedankenanstöße oder Zwischenlogik (z. B. „Zuerst filtern, dann sortieren, ...“), um den Code schrittweise aufzubauen und zu erklären.

### 4.3.1 Testfall 1: Sortierung und Ausgabe der Fallzahlen 2023

**Zielsetzung:** Die Daten des Jahres 2023 (`Fallzahlen_2023`) sollen nach der Spalte „Straftaten insgesamt“ sortiert und anschließend in einem `pandas`-DataFrame ausgegeben werden.

**Vorgehen:**

1. Einlesen der `Fallzahlen_2023`.
2. Extraktion der relevanten (Ober-)Bezirke.



- Sortierung nach **Straftaten\_insgesamt** in absteigender oder aufsteigender Reihenfolge.
- Ausgabe als **pandas DataFrame**.

**Erwartete Ausgabe:** Ein DataFrame mit mindestens folgenden Spalten:

Bezirk	Straftaten_insgesamt
(z. B. Mitte)	(z. B. 82 000)
(z. B. Neukölln)	(z. B. 50 000)
...	...

Zusätzlich können weitere Spalten (z.B. Raub, Diebstahl) enthalten sein, sofern sie nicht entfernt wurden.

#### 4.3.2 Testfall 2: Join aller Tabellen und „Bezirks-Topwert“

**Zielsetzung:** Alle DataFrames von 2014–2023 sollen *vereint* werden (Join), sodass die **Summe der Straftaten aller Jahre** pro Bezirk ermittelbar ist. Anschließend werden die Bezirke nach „den meisten Straftaten insgesamt“ sortiert ausgegeben.

**Vorgehen:**

- Einlesen und Bereinigung der einzelnen DataFrames (2014–2023).
- Zusammenführen nach dem **Bezirk**-Merkmal.
- Aggregation der „Straftaten\_insgesamt“ pro Jahr zu einer Gesamtzahl über alle Jahre.
- Ermittlung und Ausgabe der **Bezirke** sortiert nach den Summenwerten.

**Erwartete Ausgabe:** Ein DataFrame mit mindestens folgenden Spalten:

Bezirk	Straftaten_insgesamt 2014-2023
(z. B. Mitte)	(z. B. 820 000)
(z. B. Neukölln)	(z. B. 560 000)
...	...

#### 4.3.3 Testfall 3: Prozentuale Verteilung der Straftaten

**Zielsetzung:** Für ein ausgewähltes Jahr (etwa 2023) soll ermittelt werden, welcher Anteil aller Berliner Straftaten auf die jeweiligen Bezirke entfällt.

**Vorgehen:**

- Einlesen des relevanten Sheets (z. B. **Fallzahlen\_2023**).
- Berechnung des Anteils pro Bezirk:

$$\text{Prozent} = \frac{\text{Straftaten\_insgesamt pro Bezirk}}{\text{Straftaten\_Gesamtsumme}} \times 100$$

3. Ausgabe als DataFrame mit Spalten wie „Bezirk“, „Straftaten\_insgesamt“, „Anteil\_%“.

**Erwartete Ausgabe:** Ein DataFrame, bei dem jede Zeile einen **Bezirk** darstellt und mindestens folgende Spalten beinhaltet:

Bezirk	Straftaten_insgesamt	Anteil_(%)
Mitte	82 000	(z. B. 24,1%)
Neukölln	50 000	(z. B. 14,7%)
...	...	...

#### 4.3.4 Testfall 4: Zeitreihe über die Jahre 2014–2023

**Zielsetzung:** Ausgabe der **prozentualen Entwicklung** (bezogen auf *Berlin insgesamt*) der **Straftaten\_insgesamt** pro Jahr, bezogen auf das Vorjahr. Damit soll ersichtlich werden, wie sich das Gesamtaufkommen an Straftaten im Zeitverlauf verändert hat.

**Vorgehen:**

1. Einlesen sämtlicher Jahres-Sheets.
2. Addition sämtlicher Bezirkswerte für jedes Jahr, um die Gesamtzahl an Straftaten je Jahr zu erhalten.
3. Direkter Vergleich „prozentuale Änderung“ zum Vorjahr.
4. Ausgabe eines **pandas** DataFrames als Zeitreihe.

**Erwartete Ausgabe:** Ein DataFrame mit mindestens zwei Spalten:

Jahr	Straftaten_Veränderung_(%) zu Vorjahr
2014	0% (Basiswert)
2015	+3,5%
2016	-1,2%
...	...

#### 4.3.5 Testfall 5: Ermittlung des Unterbezirks mit den meisten Raubdelikten pro Oberbezirk

**Zielsetzung:** Für das Jahr 2023 soll für jeden Oberbezirk derjenige Unterbezirk ermittelt werden, der die höchste Anzahl an Raubdelikten aufweist. Dies ermöglicht eine detailliertere Analyse der Kriminalitätsverteilung auf Bezirksebene.

**Vorgehen:**

1. Einlesen des Sheets für das Jahr 2023 der Excel-Datei mit allen Unterbezirken.
2. Gruppierung der Daten nach Oberbezirk.

3. Bestimmung des Unterbezirks mit dem höchsten Wert in der Spalte **Raub** für jeden Oberbezirk.
4. Ausgabe einer Tabelle mit den Spalten: Oberbezirk, Unterbezirk, Anzahl der Raubdelikte.

**Erwartete Ausgabe:** Ein Pandas-DataFrame mit folgender Struktur:

Oberbezirk	Unterbezirk	Anzahl Raubdelikte
Mitte	Alexanderplatz	250
Pankow	Prenzlauer Berg Zentrum	180
Neukölln	Sonnenallee	220

#### 4.3.6 Testfall 6: Entwicklung der gefährlichsten Unterbezirke über die Jahre

**Zielsetzung:** Identifikation der Unterbezirke mit den höchsten Straftaten insgesamt über einen Zeitraum von mehreren Jahren. Dadurch kann analysiert werden, welche Stadtteile langfristig besonders viele Straftaten aufweisen.

**Vorgehen:**

1. Einlesen aller Jahres-Sheets über die Fallzahlen (2014–2023).
2. Aggregation der Straftaten für jeden Unterbezirk über alle Jahre hinweg.
3. Sortierung der Unterbezirke nach der höchsten Gesamtzahl an Straftaten.
4. Ausgabe einer Tabelle mit den Top 10 Unterbezirken mit den höchsten Straftaten über alle Jahre hinweg.

**Erwartete Ausgabe:** Ein Pandas-DataFrame mit folgender Struktur:

Rang	Unterbezirk	Gesamtzahl Straftaten (2014–2023)
1	Alexanderplatz	85.000
2	Kottbusser Tor	75.000
3	Sonnenallee	72.000

## 5 Methodik

### 5.1 Vorgehensweise der Untersuchung

In der Untersuchung soll geprüft werden, inwieweit Large Language Models in der Lage sind gängige Datenanalyse-Schritte auf Grundlage eines gegebenen Datensatzes durchzuführen. Hierbei wird ChatGPT als aktueller Marktführer mit dem Sprachmodell GPT-4o-mini, welches das neueste Modell ist, verwendet. Ebenso gilt es herauszufinden, wie qualitativ und effizient diese Lösung ist. Hierbei bezieht es sich auf die Forschungsfragen aus Kapitel 1.1. Für die Vorgehensweise hierbei wird zuerst der verwendete Datensatz von Berlin Open Data, wie in Kapitel 4 beschrieben, heruntergebrochen und dann an das Modell übergeben und

dazu, zu jeder verwendeten Prompting Strategie, eine Prompt verfasst. Diese Prompts können in Kapitel 5.2 eingesehen werden. Im Anschluss wird der Code ausgeführt, wobei seine Performanz gemessen wird, und es werden manuelle Analysen durchgeführt um die Qualität und Effizienz des generierten Codes zu bewerten. Die genauen Auswertungskriterien sind in Kapitel 5.3 aufgeführt. Die Ergebnisse der Auswertung werden in Kapitel 6 detailliert dargestellt.

## 5.2 Testfälle der Datenanalyse

### 5.2.1 Testfall 1

Im ersten Testfall soll der Datensatz nach einer gewissen Spalte sortiert werden. Die Begründung hierfür ist, dass dies eine sehr einfache, aber auch sehr häufig aufkommende Datenanalyse-Aufgabe ist und somit einen guten Einstieg in die Untersuchung darstellt. Die Prompts für diese Aufgabe lauten:

- **Zero-Shot Prompting:** *Ich habe eine Excel Datei mit dem Namen 'Fallzahlen.xlsx'. Hier ist der Inhalt des Sheets 'Fallzahlen\_2023': [DataFrame]. Erstelle mir ein Skript in Python, das die Daten aus der Excel-Datei einliest, nach den Straftaten insgesamt der Bezirke sortiert und in einem Dataframe speichert.*
- **Instruction Prompting:** *Ich habe eine Excel Datei mit dem Namen 'Fallzahlen.xlsx'. Hier ist der Inhalt des Sheets 'Fallzahlen\_2023': [DataFrame]. Erstelle mir ein Skript in Python, das die Daten aus der Excel-Datei einliest, nach der Spalte 'Straftaten\_insgesamt' der Bezirke sortiert und in einem Pandas Dataframe speichert. Die Zeilen mit den LOR-Schlüsseln 999900 und 999999 sollen bei der Sortierung außer Acht gelassen werden, da es sich bei diesen nicht um Bezirke handelt.*
- **Chain-of-Thought Prompting:** *Ich habe eine Excel Datei mit dem Namen 'Fallzahlen.xlsx'. Der Inhalt des Sheets ist als pandas DataFrame [DataFrame] gegeben. Bitte erstelle mir ein Python-Skript, das die folgenden Schritte ausführt:*
  - 1. Lies die Daten des Sheets 'Fallzahlen\_2023' der Excel-Datei 'Fallzahlen.xlsx' ein.*
  - 2. Sortiere die Daten nach der Spalte 'Straftaten\_insgesamt' absteigend. Zur Sortierung sollen die Zeilen mit den LOR-Schlüsseln 999900 und 999999 nicht beachtet werden, da es sich bei diesen nicht um Bezirke handelt. Sie sollen aber am Ende des Dataframes stehen bleiben.*
  - 3. Speichere das Ergebnis der Sortierung in einem Pandas Dataframe ab.**Achte darauf, dass das Skript robust ist und potentielle Fehler, wie fehlende Spalten berücksichtigt.*

### 5.2.2 Testfall 2 Verbund und Aggregation

Für den zweiten Testfall sollen die Tabellen der Excel Datei durch einen Join zusammengeführt und dann die Bezirke nach den Straftaten insgesamt von allen Jahren kombiniert geliefert werden. Die Prompts für diese Aufgabe lauten:

- **Zero-Shot Prompting:** *Ich habe eine Excel Datei mit dem Namen 'Fallzahlen.xlsx'. Erstelle mir ein Python Skript, das die Daten aller Sheets zusammenliest, sie nach der Anzahl der Straftaten insgesamt pro Bezirk sortiert und in einem Pandas DataFrame speichert. Hier sind die Daten eines der Sheets als Beispiel: [DataFrame]*
- **Instruction Prompting:** *Ich habe eine Excel Datei mit dem Namen 'Fallzahlen.xlsx'. Erstelle mir ein Python Skript, das die Daten der einzelnen Bezirke (Zeilen) aller Sheets mit einem Join zusammenfügt, sie nach der akkumulierten Spalte 'Straftaten\_insgesamt' pro Bezirk sortiert und in einem Pandas DataFrame speichert. Die Zeilen mit den LOR-Schlüsseln 999900 und 999999 sollen bei der Sortierung nicht beachtet werden, da es sich hierbei nicht um Bezirke handelt. Hier sind die Daten eines der Sheets als Beispiel: [DataFrame]*
- **Chain-of-Thought Prompting:** *Ich habe eine Excel Datei mit dem Namen 'Fallzahlen.xlsx'. Erstelle mir ein Python Skript, das folgende Anforderungen erfüllen soll:*
  - 1. Die Excel-Datei einlesen und die Sheets als DataFrames speichern.*
  - 2. Die DataFrames der einzelnen Sheets zusammen joinen, sodass pro Zeile (jede Zeile ist ein eigener Bezirk) der akkumulierte Wert der einzelnen Straftaten steht.*
  - 3. Das neue gejointe DataFrame nach der Spalte 'Straftaten\_insgesamt' sortieren. Für die Sortierung sollen die Zeilen mit den LOR-Schlüsseln 999900 und 999999 nicht beachtet werden, da es sich hierbei nicht um Bezirke handelt. Sie sollen aber am Ende des DataFrames stehen bleiben.*
  - 4. Das sortierte Pandas DataFrame zurückgeben.**Hier ist der Inhalt eines der Sheets als Beispiel: [DataFrame]*

### 5.2.3 Testfall 3

Im dritten Testfall soll das Sprachmodell die prozentualen Anteile der gesamten Straftaten der Bezirke von ganz Berlin berechnen. Die Prompts für diese Aufgabe lauten:

- **Zero-Shot Prompting:** *Ich habe eine Excel Datei mit dem Namen 'Fallzahlen.xlsx'. Erstelle mir ein Python Skript, welches den prozentualen Anteil der gesamten Straftaten der einzelnen Bezirke von den gesamten Straftaten von ganz Berlin berechnet. Hier ist der Inhalt des Sheets 'Fallzahlen\_2023': [DataFrame]*
- **Instruction Prompting:** *Ich habe eine Excel Datei mit dem Namen 'Fallzahlen.xlsx'. Erstelle mir ein Python Skript, welches den prozentualen Anteil der einzelnen Bezirke von ganz Berlin für die Spalte 'Straftaten\_insgesamt' berechnet. Jede Zeile der Tabelle ist ein einzelner Bezirk und 'Berlin (PKS gesamt)' ist die Gesamtanzahl von ganz Berlin. Hier ist der Inhalt des Sheets 'Fallzahlen\_2023': [DataFrame]*
- **Chain-of-Thought Prompting:** *Ich habe eine Excel Datei mit dem Namen 'Fallzahlen.xlsx'. Erstelle mir ein Python Skript, welches folgende Anforderungen erfüllt:*
  - 1. Die Excel-Datei einlesen*
  - 2. Die Tabelle als Pandas DataFrame speichert*

3. Überprüfen, ob die notwendigen Spalten 'Bezirke' und 'Straftaten\_insgesamt' vorhanden sind
  4. Finde die Gesamtzahl der Straftaten für ganz Berlin in der Zeile mit dem Bezirk 'Berlin (PKS gesamt)'
  5. Berechne den prozentualen Anteil der einzelnen Bezirke von ganz Berlin für die Spalte 'Straftaten\_insgesamt'
  6. Das Ergebnis als DataFrame zurückgeben
- Hier ist der Inhalt des Sheets 'Fallzahlen\_2023': `[DataFrame]`

#### 5.2.4 Testfall 4

Im vierten Testfall sollen die Skripte eine Zeitreihe der prozentualen Veränderung der gesamten Straftaten von ganz Berlin als Pandas Dataframe erstellen. Die Prompts für diese Aufgabe lauten:

- **Zero-Shot Prompting:** *Ich habe eine Excel Datei mit dem Namen 'Fallzahlen.xlsx'. Erstelle mir ein Python Skript, das die Daten aller Sheets analysiert und eine Zeitreihe mit der prozentualen Veränderung zum jeweiligen Vorjahr der gesamten Straftaten von ganz Berlin als Pandas Dataframe erstellt.*  
*Hier sind die Daten eines der Sheets als Beispiel: `[DataFrame]`*
- **Instruction Prompting:** *Ich habe eine Excel Datei mit dem Namen 'Fallzahlen.xlsx'. Erstelle mir ein Python Skript, das die Daten aller Sheets analysiert und eine Zeitreihe mit der prozentualen Veränderung der Spalte SStraftaten\_insgesamt zum jeweiligen Vorjahr von der Zeile "Berlin (PKS gesamt)" als Pandas Dataframe erstellt. Die Sheets folgen der Namensgebung 'Fallzahlen\_2014', 'Fallzahlen\_2015', etc.*  
*Hier sind die Daten eines der Sheets als Beispiel: `[DataFrame]`*
- **Chain-of-Thought Prompting:** *Ich habe eine Excel-Datei mit dem Namen 'Fallzahlen.xlsx'. Diese Datei enthält mehrere Sheets, die nach dem Muster 'Fallzahlen\_2014', 'Fallzahlen\_2015', usw. benannt sind. Jedes Sheet enthält Daten, darunter eine Spalte namens 'Straftaten\_insgesamt'. Erstelle mir ein Python Skript mit den folgenden Schritten:*
  1. *Lese alle Sheets der Excel-Datei ein und speichere jedes Sheet in einem separaten Pandas DataFrame.*
  2. *Extrahiere den Wert der Spalte 'Straftaten\_insgesamt' für die Zeile 'Berlin (PKS gesamt)' aus jedem DataFrame.*
  3. *Berechne die prozentuale Veränderung des Werts 'Straftaten\_insgesamt' zum jeweiligen Vorjahr.*
  4. *Speichere die Ergebnisse in einem neuen Pandas DataFrame, das die Jahre und die prozentuale Veränderung enthält.*

*Hier sind die Daten eines der Sheets als Beispiel: `[DataFrame]`*

### 5.2.5 Testfall 5

Im fünften Testfall sollen die Unterbezirke mit den meisten Raubdelikten pro Oberbezirk für das Jahr 2023 ermittelt werden. Die Prompts für diese Aufgabe lauten:

- **Zero-Shot Prompting:** *Ich habe eine Excel Datei mit dem Namen 'Fallzahlen&HZ 2014-2023.xlsx'. Erstelle mir ein Python Skript, das die Daten des Sheets 'Fallzahlen\_2023' ausliest und für jeden Oberbezirk den Unterbezirk mit den meisten Raubdelikten ausgibt. Hier sind die Daten des Sheets zur Orientierung: [DataFrame]*
- **Instruction Prompting:** *Ich habe eine Excel Datei mit dem Namen 'Fallzahlen&HZ 2014-2023.xlsx'. Erstelle mir ein Python Skript, das die Daten des Sheets 'Fallzahlen\_2023' ausliest und für jeden Oberbezirk den Unterbezirk mit den meisten Raubdelikten (Spaltenname:Raub) ausgibt. Hier sind die Daten des Sheets zur Orientierung: [DataFrame] Die Oberbezirke sind die Zeilen mit folgendem in der Spalte 'Bezeichnung (Bezirksregion)': Mitte, Friedrichshain-Kreuzberg, Pankow, Charlottenburg-Wilmersdorf, Spandau, Steglitz-Zehlendorf, Tempelhof-Schöneberg, Neukölln, Treptow-Köpenick, Marzahn-Hellersdorf, Lichtenberg, Reinickendorf. Die Unterbezirke sind jeweils die darunterliegenden Zeilen, bis zum nächsten Oberbezirk. Die letzten zwei Zeilen der Tabelle sind Gesamtwerte und sollen ignoriert werden.*
- **Chain-of-Thought Prompting:** *Ich habe eine Excel Datei mit dem Namen 'Fallzahlen&HZ 2014-2023.xlsx'. Hier sind die Daten des Sheets zur Orientierung: [DataFrame] Erstelle mir ein Python Skript, das folgende Schritte befolgt: 1. Die Daten des Sheets 'Fallzahlen\_2023' auslesen 2. Die Daten in einem Pandas Dataframe speichern 3. Das Dataframe auf die notwendigen Spalten reduzieren: 'Bezeichnung (Bezirksregion)' und 'Raub' 4. Unnötige Zeilen entfernen: Die letzten zwei Zeilen der Tabelle sind Gesamtwerte und sollen entfernt werden. 5. Die Oberbezirke in der Spalte 'Bezeichnung (Bezirksregion)' identifizieren. Die Oberbezirke sind die Zeilen mit folgenden Werten in der Spalte 'Bezeichnung (Bezirksregion)': Mitte, Friedrichshain-Kreuzberg, Pankow, Charlottenburg-Wilmersdorf, Spandau, Steglitz-Zehlendorf, Tempelhof-Schöneberg, Neukölln, Treptow-Köpenick, Marzahn-Hellersdorf, Lichtenberg, Reinickendorf. 6. Für jeden Oberbezirk den Unterbezirk mit dem höchsten Wert in der Spalte 'Raub' identifizieren. Die Unterbezirke sind jeweils die unter den Oberbezirken liegenden Zeilen, bis zum nächsten Oberbezirk. 7. Die Ergebnisse in einem neuen Dataframe speichern, das folgende Spalten enthält: 'Oberbezirk', 'Unterbezirk', 'Raub' 8. Das finale Dataframe ausgeben.*

### 5.2.6 Testfall 6

Im sechsten Testfall sollen die Top zehn Unterbezirke mit den meisten Straftaten über alle Jahre hinweg ermittelt werden. Die Prompts für diese Aufgabe lauten:

- **Zero-Shot Prompting:** *Ich habe eine Excel Datei mit dem Namen 'Fallzahlen&HZ 2014-2023.xlsx'. Erstelle mir ein Skript, dass die Daten aus den Sheets 'Fallzahlen\_2014' bis 'Fallzahlen\_2023' einliest die 10 Unterbezirke mit den meisten Straftaten insgesamt*

ueber alle Jahre hinweg addiert ermittelt und zurückgibt. Hier sind die Daten des Sheets 'Fallzahlen\_2023' als Beispiel: [DataFrame]

- **Instruction Prompting:** Ich habe eine Excel-Datei mit dem Namen 'Fallzahlen&HZ 2014-2023.xlsx', die die Sheets Fallzahlen\_2014 bis Fallzahlen\_2023 enthält. Erstelle ein Python-Skript, das diese Daten ausliest und in einem einheitlichen Pandas DataFrame zusammenführt. Anschließend sollen nur die Unterbezirke berücksichtigt und alle Oberbezirke sowie nicht zugeordneten Zeilen entfernt werden. Die Oberbezirke lassen sich anhand folgender Werte in der Spalte 'Bezeichnung (Bezirksregion)' identifizieren: Mitte, Friedrichshain-Kreuzberg, Pankow, Charlottenburg-Wilmersdorf, Spandau, Steglitz-Zehlendorf, Tempelhof-Schöneberg, Neukölln, Treptow-Köpenick, Marzahn-Hellersdorf, Lichtenberg, Reinickendorf. Für die verbleibenden Unterbezirke soll die Gesamtanzahl der Spalte 'Straftaten \n-insgesamt-' über alle Jahre hinweg aufsummiert und anschließend die Top 10 Unterbezirke mit den meisten Straftaten ermittelt werden. Der finale DataFrame soll nur die Spalten 'Bezeichnung (Bezirksregion)' (Unterbezirke) und die aggregierte Anzahl der Spalte 'Straftaten \n-insgesamt-' enthalten. Hier sind die Daten des Sheets Fallzahlen\_2023 zur Orientierung: [DataFrame]

- **Chain-of-Thought Prompting:**

Achte bei der Umsetzung genau auf die genannten Schritte, Spaltennamen und Sheetnamen. Hier sind die Daten des Sheets 'Fallzahlen\_2023' zur Orientierung: DataFrame Ich habe eine Excel-Datei mit dem Namen 'Fallzahlen&HZ 2014-2023.xlsx'. Diese Datei enthält unter anderem die Sheets 'Fallzahlen\_2014' bis 'Fallzahlen\_2023'. Erstelle ein Python-Skript, das die folgenden Schritte ausführt: 1. Lese alle Sheets der Excel-Datei ein und speichere jedes Sheet in einem separaten Pandas DataFrame. 2. Füge alle DataFrames zu einem einzigen zusammen, sodass ein einheitlicher DataFrame entsteht, der alle Jahre umfasst. 3. Identifiziere die Oberbezirke anhand der folgenden Werte in der Spalte 'Bezeichnung (Bezirksregion)': Mitte, Friedrichshain-Kreuzberg, Pankow, Charlottenburg-Wilmersdorf, Spandau, Steglitz-Zehlendorf, Tempelhof-Schöneberg, Neukölln, Treptow-Köpenick, Marzahn-Hellersdorf, Lichtenberg, Reinickendorf. 4. Entferne alle Zeilen, die Oberbezirke, Berlin (PKS gesamt) oder Stadt Berlin nicht zuzuordnende Einträge enthalten, sodass nur Unterbezirke übrig bleiben. 5. Summiere für jeden Unterbezirk die Werte der Spalte 'Straftaten \n-insgesamt-' über alle Jahre hinweg auf. 6. Sortiere die Unterbezirke absteigend nach der aggregierten Anzahl an Straftaten. 7. Wähle die 10 Unterbezirke mit den höchsten Gesamtstraftaten aus. 8. Erstelle einen neuen DataFrame, der nur die Spalten 'Bezeichnung (Bezirksregion)' (Unterbezirke) und die aggregierte Anzahl der Spalte 'Straftaten \n-insgesamt-' enthält. 9. Gib den finalen DataFrame aus.

Achte bei der Umsetzung genau auf die genannten Schritte, Spaltennamen und Sheetnamen. Hier sind die Daten des Sheets 'Fallzahlen\_2023' zur Orientierung: DataFrame



### 5.3 Auswertungskriterien

Die Auswertung der generierten Python Skripte erfolgt anhand der in Kapitel 1.1 definierten Kriterien. Um die Korrektheit des Codes zu messen wird das Pass@k Verfahren verwendet, dabei steht "k" für die Anzahl der ausgeführten Versuche pro Testfall. In diesem Experiment wird sich auf k=5 pro Prompt beschränkt, um eine gute Balance zwischen Genauigkeit und Rechenzeit zu finden. Bei diesem Verfahren ergibt sich als Ergebnis ein Prozentsatz über die Anzahl der erfolgreichen Versuche. Die ausgeführten Versuche werden anschließend in erfolgreich und nicht erfolgreich unterteilt und getrennt genauer betrachtet. Um zu entscheiden, ob ein Versuch erfolgreich war, wird darauf geachtet, ob die Ausführung fehlerfrei verläuft und ob das Ergebnis wie erwartet ist. Hierbei wird strikt darauf geachtet, ob das Ergebnis genau dem gewünschten Ergebnis entspricht, demnach zählt ein Testfall als nicht erfolgreich, wenn zwar die grundlegende Anforderung erfüllt ist, jedoch auch Daten im Ergebnis stehen, die dort nicht gewünscht waren. Ebenso wird ein Testfall als nicht erfolgreich betrachtet, sollte er Fehler oder Warnungen ausgeben, auch wenn dennoch ein richtiges Ergebnis dabei rauskommt. Im Code wird dabei darauf geschaut, welche Bibliotheken, Funktionen und Pandas Dataframes benutzt wurden. In der genaueren Analyse des Codes wird bei den nicht erfolgreichen Versuchen untersucht, warum der Code nicht korrekt ausgeführt wurde und was für Verbesserungen vorgenommen werden können. Bei den erfolgreichen Versuchen hingegen wird analysiert, wie der Code strukturiert ist, ob er gut dokumentiert ist, ob er erweiterbar ist und wie die Laufzeit und Ressourcennutzung des Codes abschneidet.

### 5.4 Verwendete Tools und Daten

1. **Large Language Model:** Als Large Language Model wird ChatGPT mit GPTo1-mini verwendet, da ChatGPT als aktueller Marktführer gilt und GPTo1-mini das neueste und leistungsfähigste Modell ist, welches mit der OpenAI API verfügbar ist.
2. **Libraries:** Für die Datenanalyse wird die Python-Bibliothek `pandas` verwendet, um die Excel-Dateien einzulesen und zu verarbeiten.
3. **Datenquelle:** Die Daten stammen aus den Kriminalitätsstatistiken der Stadt Berlin, die auf der Plattform Berlin Open Data veröffentlicht wurden.

## 6 Auswertung der Python-Code-Generierung zur Datenanalyse durch LLMs

### 6.1 Testfall 1: Sortierung und Ausgabe der Fallzahlen 2023

Im ersten Testfall wurde ein Python-Skript generiert, das die Excel-Tabelle „Fallzahlen\_2023“ nach der Anzahl der Straftaten insgesamt der Bezirke im Jahr 2023 sortieren sollte. Hierfür wurden drei verschiedene Prompting-Strategien (Zero-Shot, Instruction Prompting, Chain of Thought) verwendet, wobei jede Strategie fünfmal ausgeführt wurde. Die wichtigsten Beobachtungen sind:

### Erfolgsquote (Pass@15):

- **Prompt 1 (Zero-Shot Prompting):** Alle fünf Ausführungen waren teils erfolgreich. Die Daten wurden zwar korrekt sortiert das Auslesen der Daten und abspeichern in einem Pandas Dataframe wurden auch wie erwartet ausgeführt, jedoch wurden die Zeilen mit den LOR-Schlüsseln 999900 und 999999 (Berlin gesamt und nicht zuzuordnende Straftaten) in die Sortierung einbezogen, was nicht der gewünschten Anforderung entsprach.
- **Prompt 2 (Instruction Prompting):** Alle fünf Ausführungen waren teils erfolgreich. Auch hier wurden die Daten korrekt sortiert und korrekt in einem Pandas Dataframe abgespeichert und auch die Zeilen mit den LOR-Schlüsseln 999900 und 999999 korrekt aus der Sortierung ausgeschlossen, jedoch wurden sie vollständig aus dem DataFrame entfernt, was ebenfalls nicht der gewünschten Anforderung entsprach, da sie am Ende des Dataframes stehen bleiben sollten.
- **Prompt 3 (Chain of Thought Prompting):** Alle fünf Ausführungen waren erfolgreich. Die Skripte haben die Daten korrekt ausgelesen, nach der Spalte „Straftaten\_insgesamt“ sortiert und abgespeichert und behielten dabei die Zeilen mit den LOR-Schlüsseln 999900 und 999999 am Ende des DataFrames, wie gewünscht.

Berücksichtigt man die Teilerfolge als fehlgeschlagen, wie zuvor beschrieben, ergibt sich ein **Pass@15 von 33%** (5 von 15 Ausführungen waren vollständig erfolgreich). Würde man die Teilerfolge jedoch als erfolgreich betrachten, ergäbe sich ein **Pass@15 von 100%** (15 von 15 Ausführungen waren teils erfolgreich).

Tabelle 1: Ergebnisse von Testfall 1

Prompting-Strategie	Erfolgreiche Ausführungen	Fehlgeschlagene Ausführungen
Zero-Shot Prompting (Prompt 1)	0 / 5 (0%)	5 / 5 (100%)
Instruction Prompting (Prompt 2)	0 / 5 (0%)	5 / 5 (100%)
Chain of Thought Prompting (Prompt 3)	5 / 5 (100%)	0 / 5 (0%)
Gesamt	5 / 15 (33% oder 100% je nach Bewertung)	10 / 15 (67%) oder 0%

### Performance und Ressourcenverbrauch:

- Die Laufzeit betrug bei allen erfolgreichen und teils erfolgreichen Skripten nur wenige Zehntelsekunden (zwischen 0,49 und 0,56 Sekunden).
- Der Speicherverbrauch bewegte sich bei rund 150,MB *Maximum Resident Set Size*, was für diesen kleinen Datensatz sehr effizient ist.
- Die CPU-Auslastung lag bei allen Ausführungen zwischen 445% und 498%, was auf eine effiziente Nutzung der verfügbaren Ressourcen hinweist.

### Codequalität und Wartbarkeit:

- **Struktur:** Die meisten Skripte bestanden aus wenigen, übersichtlichen Schritten: *Daten einlesen, sortieren, Datei abspeichern*. Die Skripte von Prompt 3 waren dabei am strukturiertesten und enthielten zusätzliche Kommentare zur Dokumentation und Schritte zur Fehlerbehandlung und Robustheit des Codes.

- **Dokumentation:** Die Skripte waren in der Regel gut kommentiert und erklärten die Schritte und die Funktionsweise. Dies war insbesondere bei den Skripten von Prompt 3 der Fall, die zusätzliche Kommentare zur Fehlerbehandlung und zur Logik des Codes enthielten.
- **Erweiterbarkeit:** Die Skripte von Prompt 3 waren am einfachsten zu erweitern, da sie bereits eine robuste Fehlerbehandlung enthielten und die Logik der Sortierung klar dokumentiert war. Die Skripte von Prompt 1 und 2 waren weniger robust, da sie wenig zur Fehlerbehandlung enthielten.

**Fazit zu Testfall 1:** Die Ergebnisse zeigen, dass die **Prompting-Strategie C (Chain of Thought)** die zuverlässigste Methode zur Generierung von Python-Code für die Sortierung eines Excel-Datensatzes ist. Diese Strategie lieferte in allen Fällen korrekte und robuste Skripte, die den Anforderungen entsprachen. Die Strategien A und B waren weniger zuverlässig, da sie entweder die Sortierung nicht komplett korrekt durchführten oder die Daten unvollständig zurückgaben.

In Bezug auf **Performance** gab es keine Probleme und alle Skripte waren effizient. Die Codequalität war bei den Skripten von Prompt 3 am höchsten, da sie besser dokumentiert und robuster waren. Für produktive Einsätze ist daher die Verwendung von **Chain of Thought**-Prompts zu empfehlen, um sicherzustellen, dass der generierte Code den Anforderungen entspricht und leicht erweitert werden kann.

## 6.2 Testfall 2: Join aller Tabellen und Bezirks-Topwert

Im zweiten Testfall wurde ein Python-Skript generiert, das die Daten aller Excel-Sheets (2014–2023) zusammenführt, die Straftaten pro Bezirk über die Jahre summiert und die Bezirke nach der Gesamtzahl der Straftaten sortiert ausgibt. Hierfür wurden drei verschiedene Prompting-Strategien (Zero-Shot, Instruction Prompting, Chain of Thought) verwendet, wobei jede Strategie fünfmal ausgeführt wurde. Die wichtigsten Beobachtungen sind:

### Erfolgsquote (Pass@15):

- **Prompt 1 (Zero-Shot Prompting):** Drei von fünf Ausführungen waren teils erfolgreich. Die Daten wurden korrekt zusammengeführt und sortiert, jedoch wurden die unerwünschten LOR-Schlüssel (999900 und 999999) nicht korrekt behandelt. Zwei Ausführungen scheiterten vollständig, da sie die Daten nicht korrekt aggregierten. Nach Analyse des Codes fällt auf, dass die Ursache dafür ein fehlendes `groupby` war.
- **Prompt 2 (Instruction Prompting):** Zwei von fünf Ausführungen waren teilweise erfolgreich. Die Ausführungen vier und fünf haben die Daten korrekt eingelesen, zusammengefügt und sortiert, haben jedoch beide die Zeilen mit den LOR-Schlüsseln 999900 und 999999 komplett entfernt, was nicht erwünscht war. Zusätzlich hat Ausführung fünf noch mehrere `SettingWithCopyWarning` ausgegeben, was auf eine fehlerhafte Verwendung einer Funktion der Pandas Bibliothek hinweist. Die anderen Ausführungen gaben komplett fehlerhafte Ergebnisse zurück.

- **Prompt 3 (Chain of Thought Prompting):** Alle Ausführungen waren komplett erfolgreich. Die Ergebnisse waren korrekt, die Daten wurden korrekt zusammengeführt, gruppiert, aggregiert und sortiert. Ebenso wurden sie fehlerfrei in einem Pandas DataFrame abgespeichert. Die unerwünschten LOR-Schlüssel wurden korrekt behandelt und am Ende des DataFrames mit den gruppierten Werten belassen.

Die Gesamterfolgsquote beträgt **33%** (5 von 15 Ausführungen waren vollständig erfolgreich).

Tabelle 2: Ergebnisse von Testfall 2

Prompting-Strategie	Erfolgreiche Ausführungen	Fehlgeschlagene Ausführungen
Zero-Shot Prompting (Prompt 1)	0 / 5 (0%)	5 / 5 (100%)
Instruction Prompting (Prompt 2)	0 / 5 (0%)	5 / 5 (100%)
Chain of Thought Prompting (Prompt 3)	5 / 5 (100%)	0 / 5 (0%)
<b>Gesamt</b>	<b>5 / 15 (33% oder 100% je nach Bewertung)</b>	<b>10 / 15 (67%) oder 0%</b>

### Performance und Ressourcenverbrauch:

- Die Laufzeit betrug bei allen Ausführungen zwischen 2,83 und 3,28 Sekunden, was für die Verarbeitung von 10 Excel-Sheets mit insgesamt 140 Zeilen sehr effizient ist.
- Der Speicherverbrauch bewegte sich bei rund 150–155 MB *Maximum Resident Set Size*, was für diesen Datensatz angemessen ist.
- Die CPU-Auslastung lag bei allen Ausführungen zwischen 157% und 168%, was auf eine effiziente Nutzung der verfügbaren Ressourcen hinweist.

### Codequalität und Wartbarkeit:

- **Struktur:** Die Struktur ist in allen drei Prompts gut unterteilt und übersichtlich. In allen Skripten waren die Schritte klar definiert und einfach nachvollziehbar.
- **Dokumentation:** Alle Skripte sind gut kommentiert und erklärten die einzelnen Schritte. In allen drei Prompts gab es einige Skripte die etwas deutlicher und ausführlicher dokumentiert waren als andere. Jedoch waren hierbei keine großen Unterschiede zwischen den Prompts zu erkennen.
- **Erweiterbarkeit:** Alle Skripte sind leicht erweiterbar und können einfach angepasst werden, um zusätzliche Funktionalitäten hinzuzufügen. Es wurden teils auch Erweiterungsmöglichkeiten für die Skripte gegeben, wie zum Beispiel das generieren einer neuen Excel-Datei, sofern dieses nicht schon im Skript selbst enthalten war. Auch die Fehler aus den fehlgeschlagenen Ausführungen sind leicht zu beheben.

**Fazit zu Testfall 2:** Die Ergebnisse zeigen, dass die **Chain of Thought**-Strategie die zuverlässigste Methode zur Generierung von Python-Code für die Zusammenführung und Sortierung von Excel-Daten ist. Diese Strategie lieferte in den meisten Fällen korrekte und robuste Skripte, die den Anforderungen entsprachen. Die **Zero-Shot**- und **Instruction Prompting**-Strategien waren weniger zuverlässig, da sie entweder die Filterung nicht korrekt durchführten oder die Daten unvollständig zurückgaben. In Bezug auf **Performance** gab es keine Probleme, und alle Skripte waren effizient.

### 6.3 Testfall 3: Prozentuale Verteilung der Straftaten

Im dritten Testfall wurde ein Python-Skript generiert, das den prozentualen Anteil der Straftaten pro Bezirk an den gesamten Straftaten in Berlin berechnet. Hierfür wurden drei verschiedene Prompting-Strategien (Zero-Shot, Instruction Prompting, Chain of Thought) verwendet, wobei jede Strategie fünfmal ausgeführt wurde. Die wichtigsten Beobachtungen sind:

#### Erfolgsquote (Pass@15):

- **Prompt 1 (Zero-Shot):** Alle fünf Ausführungen waren erfolgreich. Die Skripte berechneten die prozentualen Anteile korrekt und speicherten die Ergebnisse in einer Excel-Datei.
- **Prompt 2 (Instruction Prompting):** Alle fünf Ausführungen waren erfolgreich. Die Skripte berechneten die prozentualen Anteile korrekt und speicherten die Ergebnisse in einer Excel-Datei.
- **Prompt 3 (Chain of Thought):** Vier von fünf Ausführungen waren erfolgreich. Eine Ausführung scheiterte aufgrund eines Syntaxfehlers, bei dem zwei Python-Schlüsselwörter („if“ und „not“) ins Deutsche übersetzt wurde.

Die Gesamterfolgsquote beträgt **93%** (14 von 15 Ausführungen waren erfolgreich).

Tabelle 3: Ergebnisse von Testfall 3

Prompting-Strategie	Erfolgreiche Ausführungen	Fehlgeschlagene Ausführungen
Zero-Shot Prompting (Prompt 1)	5 / 5 (100%)	0 / 5 (0%)
Instruction Prompting (Prompt 2)	5 / 5 (100%)	0 / 5 (0%)
Chain of Thought Prompting (Prompt 3)	4 / 5 (80%)	1 / 5 (20%)
<b>Gesamt</b>	<b>14 / 15 (93%)</b>	<b>1 / 15 (7%)</b>

#### Performance und Ressourcenverbrauch:

- Die Laufzeit betrug bei allen erfolgreichen Ausführungen zwischen 0,51 und 0,59 Sekunden, was für die Berechnung der prozentualen Anteile sehr effizient ist.
- Der Speicherverbrauch bewegte sich bei rund 150–155 MB *Maximum Resident Set Size*, was für diesen Datensatz angemessen ist.
- Die CPU-Auslastung lag bei allen Ausführungen zwischen 424% und 479%, was auf eine effiziente Nutzung der verfügbaren Ressourcen hinweist.

#### Codequalität und Wartbarkeit:

- **Struktur:** Die Struktur ist in allen drei Prompts gut unterteilt und übersichtlich. In allen Skripten waren die Schritte klar definiert und einfach nachvollziehbar.
- **Dokumentation:** Alle Skripte sind gut kommentiert und erklärten die einzelnen Schritte. Bei Prompt drei waren die Skripte etwas deutlicher und ausführlicher dokumentiert als bei den anderen beiden Prompts.

- **Erweiterbarkeit:** Alle Skripte sind leicht erweiterbar und können einfach angepasst werden, um zusätzliche Funktionalitäten hinzuzufügen. Es wurden teils auch Erweiterungsmöglichkeiten für die Skripte gegeben, wie zum Beispiel das generieren einer neuen Excel-Datei, sofern dieses nicht schon im Skript selbst enthalten war.

**Fazit zu Testfall 3:** Die Ergebnisse zeigen, dass die **Zero-Shot**- und **Instruction Prompting**-Strategien in allen Fällen erfolgreich waren, während die **Chain of Thought (CoT)**-Strategie aufgrund eines Syntaxfehlers in einer Ausführung scheiterte. Jedoch wurden hier die zwei Python-Schlüsselwörter „if“ und „not“ ins Deutsche übersetzt, was zu einem Syntaxfehler führte. Es ist unklar, ob dies darauf zurückzuführen ist, dass die Beschreibung der Aufgabenstellung zu lang war und das Modell dadurch verwirrt wurde, oder ob es sich um einen Zufallsfehler handelt.

In Bezug auf **Performance** gab es keine Probleme, und alle Skripte waren effizient. Für Aufgaben dieses Umfangs lässt sich daher keine signifikante Verbesserung der Performance durch die Wahl einer bestimmten Prompting-Strategie feststellen.

#### 6.4 Testfall 4: Zeitreihe über die Jahre 2014–2023

Im vierten Testfall wurde ein Python-Skript generiert, das die prozentuale Veränderung der Straftaten in Berlin im Vergleich zum Vorjahr berechnet und als Zeitreihe ausgibt. Hierfür wurden drei verschiedene Prompting-Strategien (Zero-Shot, Instruction Prompting, Chain of Thought) verwendet, wobei jede Strategie fünfmal ausgeführt wurde. Die wichtigsten Beobachtungen sind:

##### Erfolgsquote (Pass@15):

- **Prompt 1 (Zero-Shot):** Zwei von fünf Ausführungen waren erfolgreich. Die erfolgreichen Ausführungen berechneten die prozentuale Veränderung korrekt. Drei Ausführungen scheiterten: zwei aufgrund von Problemen mit den Sheet-Namen und eine aufgrund eines Syntaxfehlers, bei dem ein Python-Schlüsselwort („not“) ins Deutsche übersetzt wurde.
- **Prompt 2 (Instruction Prompting):** Alle fünf Ausführungen waren erfolgreich. Die Skripte berechneten die prozentuale Veränderung korrekt und speicherten die Ergebnisse in einem DataFrame.
- **Prompt 3 (Chain of Thought):** Vier von fünf Ausführungen waren erfolgreich. Eine Ausführung lieferte zwar die richtigen Ergebnisse, warf jedoch eine Warnung aufgrund einer veralteten Pandas-Funktionalität.

Die Gesamterfolgsquote beträgt **73%** (11 von 15 Ausführungen waren erfolgreich).

##### Performance und Ressourcenverbrauch:

- Die Laufzeit betrug bei allen erfolgreichen Ausführungen zwischen 2,85 und 3,47 Sekunden, was für die Berechnung der prozentualen Veränderung sehr effizient ist.

Tabelle 4: Ergebnisse von Testfall 4

Prompting-Strategie	Erfolgreiche Ausführungen	Fehlgeschlagene Ausführungen
Zero-Shot Prompting (Prompt 1)	2 / 5 (40%)	3 / 5 (60%)
Instruction Prompting (Prompt 2)	5 / 5 (100%)	0 / 5 (0%)
Chain of Thought Prompting (Prompt 3)	4 / 5 (80%)	1 / 5 (20%)
<b>Gesamt</b>	<b>11 / 15 (73%)</b>	<b>4 / 15 (27%)</b>

- Der Speicherverbrauch bewegte sich bei rund 150–160 MB *Maximum Resident Set Size*, was für diesen Datensatz angemessen ist.
- Die CPU-Auslastung lag bei allen Ausführungen zwischen 155% und 168%, was auf eine effiziente Nutzung der verfügbaren Ressourcen hinweist.

#### Codequalität und Wartbarkeit:

- **Struktur:** Die Struktur ist in allen drei Prompts gut unterteilt und übersichtlich. In allen Skripten waren die Schritte klar definiert und einfach nachvollziehbar.
- **Dokumentation:** Alle Skripte sind gut kommentiert und erklärten die einzelnen Schritte. In diesem Testfall fiel auf, dass die Skripte der **Chain of Thought**-Prompts nicht die am besten dokumentierten waren, sondern Prompt zwei die meisten Skripte mit einer besonders ausführlichen Dokumentation hatte.
- **Erweiterbarkeit:** Alle Skripte sind leicht erweiterbar und können einfach angepasst werden, um zusätzliche Funktionalitäten hinzuzufügen. Es wurden teils auch Erweiterungsmöglichkeiten für die Skripte gegeben, wie zum Beispiel das generieren einer neuen Excel-Datei, sofern dieses nicht schon im Skript selbst enthalten war.

**Fazit zu Testfall 4:** Die Ergebnisse zeigen, dass die **Instruction Prompting**- und **Chain of Thought**-Strategien die besten Ergebnisse liefern, während die **Zero-Shot**-Strategie aufgrund von Problemen mit den Sheet-Namen und Syntaxfehlern in drei von fünf Ausführungen scheiterte. Anders als bei den vorherigen Testfällen war die **Chain of Thought**-Strategie nicht die zuverlässigste, da sie in einer Ausführung eine Warnung aufgrund einer veralteten Pandas-Funktionalität ausgab. Stattdessen war die **Instruction Prompting**-Strategie die am besten zu bewertende Prompting-Strategie.

In Bezug auf **Performance** gab es keine Probleme, und alle Skripte waren effizient. Die Codequalität war bei den Skripten von Prompt 2 am höchsten, da sie gut dokumentiert und robust waren.

## 6.5 Testfall 5: Ermittlung des Unterbezirks mit den meisten Raubdelikten pro Oberbezirk

Im fünften Testfall wurde ein Python-Skript generiert, das für jeden Oberbezirk den Unterbezirk mit den meisten Raubdelikten ermittelt. Es wurden drei verschiedene Prompting-Strategien (Zero-Shot, Instruction-based, Chain of Thought) verwendet, wobei jede Strategie fünfmal ausgeführt wurde.

#### Erfolgsquote (Pass@5):

- **Zero-Shot Prompting:** Nur ein von fünf Versuchen war erfolgreich (**20 %**). Hauptprobleme waren falsche Spaltennamen und fehlerhafte Gruppierung.
- **Instruction-based Prompting:** Alle fünf Ausführungen waren erfolgreich (**100 %**).
- **Chain of Thought Prompting:** Alle fünf Ausführungen waren erfolgreich (**100 %**).

Tabelle 5: Erfolgsquote der generierten Skripte

Prompting-Strategie	Erfolgreiche Ausführungen	Fehlgeschlagene Ausführungen
Zero-Shot Prompting	1 / 5 (20%)	4 / 5 (80%)
Instruction-based Prompting	5 / 5 (100%)	0 / 5 (0%)
Chain of Thought Prompting	5 / 5 (100%)	0 / 5 (0%)
<b>Gesamt</b>	<b>11 / 15 (73%)</b>	<b>4 / 15 (27%)</b>

#### Codequalität und Performance:

- Die durchschnittliche Laufzeit lag bei **0,55 Sekunden**.
- CPU-Auslastung lag zwischen **420 %** und **460 %**.
- Die besten Ergebnisse erzielten **Instruction-based** und **Chain of Thought Prompting**.

**Fazit zu Testfall 5:** Instruction-based und Chain of Thought Prompting sind die besten Methoden für diesen Anwendungsfall, während Zero-Shot Prompting nicht empfohlen wird.

### 6.6 Testfall 6: Entwicklung der gefährlichsten Unterbezirke über die Jahre

Im sechsten Testfall wurde ein Python-Skript generiert, das die Unterbezirke mit den höchsten Straftaten insgesamt über die Jahre 2014 bis 2023 identifiziert. Hierfür wurden drei verschiedene Prompting-Strategien (Zero-Shot, Instruction Prompting, Chain of Thought) verwendet, wobei jede Strategie fünfmal ausgeführt wurde. Die wichtigsten Beobachtungen sind:

#### Erfolgsquote (Pass@15):

- **Prompt 1 (Zero-Shot):** Keine der fünf Ausführungen war erfolgreich. Alle Skripte scheiterten daran, dass die erwarteten Spalten in den Excel-Sheets nicht vorhanden waren, was zu Fehlern beim Zugriff auf die Daten führte.
- **Prompt 2 (Instruction Prompting):** Keine der fünf Ausführungen war erfolgreich. Die Skripte stießen durchweg auf Probleme mit der Spaltenbenennung ('KeyError: 'Straftaten -insgesamt-'), was dazu führte, dass keine brauchbaren Ergebnisse generiert wurden.



- **Prompt 3 (Chain of Thought):** Drei von fünf Ausführungen waren erfolgreich. Die erfolgreichen Skripte lieferten korrekte Ergebnisse für die zehn Unterbezirke mit den meisten Straftaten. Die beiden fehlgeschlagenen Ausführungen scheiterten aufgrund falscher Spaltenbenennung oder Problemen beim Laden der Excel-Datei.

Die Gesamterfolgsquote beträgt **20%** (3 von 15 Ausführungen waren erfolgreich).

Tabelle 6: Ergebnisse von Testfall 6

Prompting-Strategie	Erfolgreiche Ausführungen	Fehlgeschlagene Ausführungen
Zero-Shot Prompting (Prompt 1)	0 / 5 (0%)	5 / 5 (100%)
Instruction Prompting (Prompt 2)	0 / 5 (0%)	5 / 5 (100%)
Chain of Thought Prompting (Prompt 3)	3 / 5 (60%)	2 / 5 (40%)
<b>Gesamt</b>	<b>3 / 15 (20%)</b>	<b>12 / 15 (80%)</b>

#### Performance und Ressourcenverbrauch:

- Die Laufzeit der erfolgreichen Skripte lag zwischen 3,38 und 3,70 Sekunden, was für die Berechnung von aggregierten Straftaten über zehn Jahre hinweg effizient ist.
- Der Speicherverbrauch variierte zwischen 156–160 MB *Maximum Resident Set Size*, was für diesen Datensatz als angemessen betrachtet werden kann.
- Die CPU-Auslastung schwankte zwischen 150% und 156%, was auf eine effiziente Verarbeitung der Daten hinweist.

#### Codequalität und Wartbarkeit:

- **Struktur:** Die erfolgreichen Skripte waren gut strukturiert und in sinnvolle Abschnitte unterteilt. Sie führten die Schritte zur Datenaggregation in einer klar nachvollziehbaren Reihenfolge durch.
- **Dokumentation:** Die Chain of Thought-Prompts enthielten in den erfolgreichen Skripten eine gute Dokumentation. Im Gegensatz dazu fehlte in den anderen Strategien oft eine klare Kommentierung der einzelnen Verarbeitungsschritte.
- **Erweiterbarkeit:** Die erfolgreichen Skripte waren flexibel und könnten leicht erweitert werden, z.B. durch zusätzliche Filter oder eine visuelle Darstellung der Daten in Form von Diagrammen.

**Fazit zu Testfall 6:** Die Ergebnisse zeigen, dass die **Chain of Thought**-Strategie die besten Ergebnisse erzielte, da drei der fünf Ausführungen erfolgreich waren. Die **Zero-Shot**-Strategie und die **Instruction Prompting**-Strategie scheiterten in allen fünf Durchläufen, da sie nicht in der Lage waren, die korrekten Spaltennamen zu erkennen und entsprechend zu verarbeiten.

In Bezug auf **Performance** gab es keine auffälligen Probleme, und alle erfolgreichen Skripte lieferten Ergebnisse in einem angemessenen Zeitrahmen. Die **Codequalität** war insbesondere in den erfolgreichen Chain of Thought-Skripten hoch, während die weniger erfolgreichen Prompts oft an unzureichender Fehlerbehandlung und mangelnder Dokumentation litten.

Insgesamt zeigt sich, dass für diesen Testfall eine detaillierte und schrittweise Herangehensweise (Chain of Thought) die zuverlässigsten Ergebnisse liefert. Ein robustes Handling von fehlerhaften oder unerwarteten Spaltennamen ist dabei entscheidend, um die Erfolgsquote zu verbessern.

## 6.7 Übersicht der Ergebnisse

Tabelle 7: Übersicht der Pass@15-Ergebnisse pro Testfall und Prompting-Strategie (Es wird nur die Erfolgsquote der komplett korrekten Ergebnisse angegeben)

Testfall	Zero-Shot	Instruction	Chain of Thought
1	0%	0%	100%
2	0%	0%	100%
3	100%	100%	80%
4	40%	100%	80%
5	20%	100%	100%
6	0%	0%	60%

## 6.8 Vergleich manuell erstellter Code

In dieser Untersuchung werden die durch Large Language Models (LLMs) generierten Python-Skripte mit manuell erstellten Skripten verglichen. Dabei liegt der Fokus auf der Korrektheit der Resultate, der Code-Qualität sowie der Wartbarkeit und Erweiterbarkeit der Lösungen. Ziel ist es, Unterschiede und Gemeinsamkeiten der beiden Ansätze herauszuarbeiten.

**Korrektheit der Resultate** Ein wesentlicher Aspekt ist die inhaltliche Korrektheit der generierten Skripte. Während die manuell erstellten Skripte in allen Testfällen die erwarteten Ergebnisse lieferten – was darauf zurückzuführen ist, dass sie gezielt an die spezifischen Anforderungen angepasst wurden – traten bei den LLM-generierten Skripten vereinzelt Fehler auf. Diese Fehler betrafen insbesondere die falsche oder unvollständige Verarbeitung bestimmter Daten, beispielsweise durch nicht korrekt benannte Spalten, fehlerhafte Aggregationen oder eine nicht beachtete Filterung bestimmter Zeilen (LOR-Schlüssel). Dies führte in einigen Testfällen dazu, dass das Ergebnis nicht der gewünschten Spezifikation entsprach. Daher bedarf es bei den generierten Skripten teilweise weiterer Arbeit, insbesondere wenn die Prompt unpräzise formuliert war. Dies zeigt, dass die Qualität der Prompts eine entscheidende Rolle für die Korrektheit der generierten Skripte spielt.

**Performance** Hinsichtlich der Laufzeiten und der Ressourcennutzung (z. B. CPU-Auslastung, Speicherbedarf) sind keine gravierenden Unterschiede zwischen den manuell erstellten Skripten und den generierten Skripten festzustellen. Beide Varianten führen die Analysen innerhalb weniger Sekunden aus und verbrauchen typischerweise zwischen 150 MB und 160 MB Maximum Resident Set Size. In diesem Anwendungsbereich – also bei den vorliegenden Datensätzen und Aufgaben – erweisen sich die Skripte beider Ansätze als annähernd gleichwertig in Bezug auf die Performance.

**Code-Struktur und Wartbarkeit** Aus der Perspektive von Codequalität und Wartbarkeit zeigen sich leichte Unterschiede. Die manuell geschriebenen Skripte sind in der Regel stärker auf die konkrete Aufgabenstellung zugeschnitten und enthalten genau die für die jeweilige Problemstellung nötige Logik. Bei den durch ein LLM generierten Skripten fällt hingegen auf, dass sie teils zusätzliche Codeabschnitte (z. B. zur Fehlerbehandlung) oder Kommentare bereitstellen, da das Modell versucht, einen möglichst robusten und verallgemeinerten Ansatz zu liefern. Dies kann den Quellcode einerseits etwas länger machen, andererseits erhöht es unter Umständen die Wiederverwendbarkeit und Anpassbarkeit. In Summe sind jedoch beide Varianten in klar nachvollziehbare Schritte gegliedert und somit gut wart- und erweiterbar.

## 7 Fazit und Ausblick

TODO:check Die vorliegende Arbeit hat untersucht, wie gut sich moderne *Large Language Models* (LLMs) – exemplarisch repräsentiert durch ChatGPT mit dem Modell „GPTo1“ – für die automatisierte Generierung von Python-Code im Kontext typischer Datenanalyseaufgaben eignen. Im Fokus standen vier typische Szenarien: das Sortieren und Filtern von Daten (Testfall1), das Zusammenführen und Aggregieren verschiedener Tabellen (Testfall2), die Berechnung prozentualer Anteile (Testfall3) sowie die Ermittlung von Zeitreihen mitsamt prozentualer Veränderung zum Vorjahr (Testfall4). Dabei wurden unterschiedliche Prompting-Strategien (Zero-Shot, Instruction Prompting und Chain of Thought) eingesetzt und die Qualität der resultierenden Skripte anhand von Kriterien wie Korrektheit, Performanz, Code-Struktur und Wartbarkeit evaluiert.

### Zentrale Erkenntnisse

- **Korrektheit und Erfolgsquote:**

Die Ergebnisse zeigen, dass die Leistungsfähigkeit stark von der Art des Promptings abhängt. Während *Chain of Thought*-Prompts im Durchschnitt sehr robuste und korrekte Lösungen lieferten, kam es bei Zero-Shot-Ansätzen häufiger zu Syntaxfehlern, falschen Sheet-Namen oder fehlerhafter Ausfilterung bestimmter Zeilen. Insgesamt bestätigte sich, dass *strukturierte Prompt-Anweisungen* (Instruction oder Chain of Thought) zu einer höheren Erfolgsquote führen als ein unpräziser Zero-Shot-Ansatz.

- **Performanz:**

Bei allen Testfällen war die Ausführungszeit sehr kurz (teils nur wenige Zehntelsekunden bis wenige Sekunden), was für den überschaubaren Datensatz (mehrere Excel-Sheets, aber jeweils nur im Bereich einiger Zeilen pro Sheet) vollkommen ausreichend ist. Damit bestätigt sich, dass ein KI-basierter Code-Generator im Kontext kleiner bis mittlerer Datenanalysen in Sachen Laufzeit absolut konkurrenzfähig ist.

- **Qualität und Wartbarkeit:**

Die generierten Skripte weisen – insbesondere bei Chain of Thought – eine erstaunlich gute Code-Struktur auf. Sie sind in der Regel sinnvoll kommentiert, nutzen etablierte Python-Bibliotheken (z.B. *pandas*) und sind leicht erweiterbar. Limitationen zeigen sich

jedoch in Fällen, in denen das Modell falsche oder irreführende Annahmen über Spaltennamen oder Datenstrukturen trifft. Solche Fehler entstehen meist durch Unklarheiten oder fehlende Details in den Prompts.

- **Grenzen:**

Trotz der teils hohen Erfolgsquote in den definierten Testfällen muss beachtet werden, dass in realen Szenarien Datenanalysen oft komplexere Anforderungen mit sich bringen: Ungewohnte Datenformate, aufwändige Vorverarbeitungs- und Qualitätschecks oder tiefergehende statistische Verfahren. Hier können LLMs ohne exakte Vorgaben schnell an ihre Grenzen gelangen. Die Modelle tendieren außerdem gelegentlich zu „Halluzinationen“, indem sie nicht-existente Funktionen oder Spalten vorschlagen.

**Ausblick** Die steigende Leistungsfähigkeit von LLMs lässt erwarten, dass die automatische Code-Generierung in den kommenden Jahren eine immer größere Rolle in Datenanalyse- und Data-Science-Projekten spielen wird. Insbesondere die folgenden Entwicklungen erscheinen relevant:

- **Spezialisierte Modelle und Fine-Tuning:**

Mit zunehmender Verfügbarkeit von domänenspezifischen Datensätzen für Code-Generierung könnten LLMs besser auf bestimmte Aufgaben, wie z.,B. Datenbereinigung oder automatisiertes Exploratory Data Analysis (EDA), optimiert werden. Eine engere Anbindung an unternehmenseigene Datenbanken ist ebenfalls denkbar.

- **Bessere Kontextverarbeitung:**

Aktuelle LLMs haben eine begrenzte Kontextlänge. Künftige Modelle werden voraussichtlich größere Kontextfenster bieten und so umfangreichere Code-Dateien oder Datenschemata verarbeiten können. Damit würden komplexere Use Cases (etwa umfangreiche ETL-Pipelines) in den Bereich des Möglichen rücken.

- **Integrierte Fehlersuche und Debugging:**

Erste Ansätze zeigen, dass LLMs nicht nur Code generieren, sondern auch zum Debugging eingesetzt werden können. Wenn die Modelle direkt während der Code-Erstellung mögliche Probleme erkennen und Lösungsvorschläge machen, könnte die Produktivität bei Datenanalyse-Projekten weiter gesteigert werden.

- **Erweiterte Qualitäts- und Sicherheitstests:**

Wie in aktuellen Studien – beispielsweise von Liu et al. (2023) – diskutiert, bedarf es verbesserter Evaluationsmethoden (etwa *EvalPlus*), um die tatsächliche Funktionalität und Sicherheit des generierten Codes sicherzustellen. Automatisierte Unit-Tests und Code-Audits im Prompt bzw. Post-Processing könnten ein wesentlicher Baustein sein.

Zusammenfassend lässt sich festhalten, dass LLMs bereits heute in der Lage sind, für standardisierte Datenanalyseschritte in Python zuverlässig einsatzfähigen Code zu erstellen. Dies kann den Einstieg in die Datenanalyse vereinfachen und auch erfahrenen Anwendern mühsame Routineaufgaben abnehmen. Allerdings bleibt ein gewisses Maß an menschlicher Kontrolle

weiterhin unverzichtbar: Sei es zur Prüfung potenzieller Halluzinationen, zur Anpassung spezieller Projektanforderungen oder zur Qualitätssicherung komplexer Analysen. Das Potenzial für zukünftige Anwendungen ist groß – insbesondere, wenn die Modelle durch spezialisierte Trainingsdaten, fortgeschrittene Prompting-Techniken und integrierte Debugging-Funktionen weiter verbessert werden.

## 8 Anhang

### 8.1 Literaturverzeichnis

#### Literatur

- [1] Jiawei Liu u. a. „Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation“. In: *Advances in Neural Information Processing Systems*. Hrsg. von A. Oh u. a. Bd. 36. Curran Associates, Inc., 2023, S. 21558–21572. URL: [https://proceedings.neurips.cc/paper\\_files/paper/2023/file/43e9d647ccd3e4b7b5baab53f0368686-Paper-Conference.pdf](https://proceedings.neurips.cc/paper_files/paper/2023/file/43e9d647ccd3e4b7b5baab53f0368686-Paper-Conference.pdf).
- [2] Mark Chen u. a. *Evaluating Large Language Models Trained on Code*. 2021. arXiv: 2107.03374 [cs.LG]. URL: <https://arxiv.org/abs/2107.03374>.
- [3] Jiawei Liu u. a. „Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation“. In: *Thirty-seventh Conference on Neural Information Processing Systems*. 2023. URL: <https://openreview.net/forum?id=1qv610Cu7>.
- [4] Wayne Xin Zhao u. a. *A Survey of Large Language Models*. 2024. arXiv: 2303.18223 [cs.CL]. URL: <https://arxiv.org/abs/2303.18223>.
- [5] Juyong Jiang u. a. *A Survey on Large Language Models for Code Generation*. 2024. arXiv: 2406.00515 [cs.CL]. URL: <https://arxiv.org/abs/2406.00515>.
- [6] Shihan Dou u. a. *What’s Wrong with Your Code Generated by Large Language Models? An Extensive Study*. 2024. arXiv: 2407.06153 [cs.SE]. URL: <https://arxiv.org/abs/2407.06153>.
- [7] Dong Huang u. a. *EffiBench: Benchmarking the Efficiency of Automatically Generated Code*. 2024. arXiv: 2402.02037 [cs.SE]. URL: <https://arxiv.org/abs/2402.02037>.
- [8] Pengfei Liu u. a. *Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing*. 2021. arXiv: 2107.13586 [cs.CL]. URL: <https://arxiv.org/abs/2107.13586>.
- [9] Tom B. Brown u. a. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL]. URL: <https://arxiv.org/abs/2005.14165>.
- [10] Long Ouyang u. a. *Training language models to follow instructions with human feedback*. 2022. arXiv: 2203.02155 [cs.CL]. URL: <https://arxiv.org/abs/2203.02155>.
- [11] Jason Wei u. a. *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*. 2023. arXiv: 2201.11903 [cs.CL]. URL: <https://arxiv.org/abs/2201.11903>.
- [12] Baptiste Rozière u. a. *Code Llama: Open Foundation Models for Code*. 2024. arXiv: 2308.12950 [cs.CL]. URL: <https://arxiv.org/abs/2308.12950>.
- [13] Raymond Li u. a. *StarCoder: may the source be with you!* 2023. arXiv: 2305.06161 [cs.CL]. URL: <https://arxiv.org/abs/2305.06161>.

- [14] Yue Wang u. a. *CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation*. 2021. arXiv: 2109.00859 [cs.CL]. URL: <https://arxiv.org/abs/2109.00859>.
- [15] Erik Nijkamp u. a. *CodeGen: An Open Large Language Model for Code with Multi-Turn Program Synthesis*. 2023. arXiv: 2203.13474 [cs.LG]. URL: <https://arxiv.org/abs/2203.13474>.
- [16] Rudra Dhar, Karthik Vaidhyanathan und Vasudeva Varma. *Can LLMs Generate Architectural Design Decisions? -An Exploratory Empirical study*. 2024. arXiv: 2403.01709 [cs.SE]. URL: <https://arxiv.org/abs/2403.01709>.
- [17] Jiawei Liu u. a. „Evaluating Language Models for Efficient Code Generation“. In: *First Conference on Language Modeling*. 2024. URL: <https://openreview.net/forum?id=IBCBMeAhmC>.
- [18] Polizei Berlin. *Kriminalitätsatlas Berlin*. 2024. URL: <https://daten.berlin.de/datensaetze/kriminalitaetsatlas-berlin>.

## 8.2 Quellcodeverzeichnis

### 8.2.1 Testfall 1

```

1 from openai import OpenAI
2 import os
3 import pandas as pd
4
5 path=os.getenv("THESIS_PATH")
6
7 # OpenAI API Key
8 api_key = os.getenv("OPENAI_API_KEY")
9 client = OpenAI(api_key=api_key)
10
11 # Excel-Datei einlesen und als Dataframe speichern
12 file = path+"/testcases/Fallzahlen.xlsx"
13 sheet_name = "Fallzahlen_2023"
14 df = pd.read_excel(file, sheet_name=sheet_name)
15
16 # Prompt erstellen: CSV-Daten einbetten + Anweisung
17 prompt = f"""
18 Ich habe eine Excel Datei mit dem Namen 'Fallzahlen.xlsx'. Hier ist der
19 Inhalt des Sheets 'Fallzahlen_2023': {df}. Erstelle mir ein Skript in
20 Python, das die Daten aus der Excel-Datei einliest, nach den Straftaten
21 insgesamt der Bezirke sortiert und in einem Dataframe speichert.
22 """
23
24 # Anfrage an die OpenAI Chat-API
25 response = client.chat.completions.create(model="o1-mini",
26 messages=[
27 {"role": "user", "content": prompt}
28 ],)

```

```

27 response_file = path+"testcases/testcase1/prompt1/exec5/response.txt"
28 os.makedirs(os.path.dirname(response_file), exist_ok=True)
29 # Ausgabe des generierten Skripts
30 with open(response_file, "w") as file:
31     file.write(response.choices[0].message.content)
32
33 print(f"Response wurde in {response_file} gespeichert.")

```

Listing 1: OpenAI API Request Skript

```

1 import pandas as pd
2
3 # Pfad zur Excel-Datei
4 datei_pfad = 'Fallzahlen.xlsx'
5
6 try:
7     # Einlesen des spezifischen Sheets 'Fallzahlen_2023'
8     df = pd.read_excel(datei_pfad, sheet_name='Fallzahlen_2023')
9
10    # Anzeigen der ersten paar Zeilen zur Überprüfung
11    print("Original DataFrame:")
12    print(df.head())
13
14    # Sortieren nach der Spalte 'Straftaten_insgesamt' in absteigender
    Reihenfolge
15    df_sortiert = df.sort_values(by='Straftaten_insgesamt', ascending=False)
16
17    # Zurücksetzen des Index (optional)
18    df_sortiert.reset_index(drop=True, inplace=True)
19
20    # Anzeige der sortierten DataFrame
21    print("\nSortierter DataFrame nach 'Straftaten_insgesamt':")
22    print(df_sortiert)
23
24    # Optional: Speichern des sortierten DataFrames in eine neue Excel-Datei
25    sortierte_datei_pfad = 'Fallzahlen_sortiert.xlsx'
26    df_sortiert.to_excel(sortierte_datei_pfad, index=False)
27    print(f"\nDer sortierte DataFrame wurde in '{sortierte_datei_pfad}'
    gespeichert.")
28
29 except FileNotFoundError:
30     print(f"Die Datei '{datei_pfad}' wurde nicht gefunden.")
31 except ValueError as ve:
32     print(f"Fehler beim Einlesen des Sheets: {ve}")
33 except Exception as e:
34     print(f"Ein unerwarteter Fehler ist aufgetreten: {e}")

```

Listing 2: Skript Testfall 1 Prompt 1 Ausführung 1

```

1 import pandas as pd
2
3 # Pfad zur Excel-Datei
4 excel_datei = 'Fallzahlen.xlsx'

```



```

5
6 # Name des Sheets
7 sheet_name = 'Fallzahlen_2023'
8
9 # Einlesen der Excel-Datei
10 df = pd.read_excel(excel_datei, sheet_name=sheet_name)
11
12 # Anzeigen der ersten paar Zeilen (optional)
13 print("Unsortierte Daten:")
14 print(df.head())
15
16 # Sortieren nach 'Straftaten_insgesamt' in absteigender Reihenfolge
17 df_sortiert = df.sort_values(by='Straftaten_insgesamt', ascending=False)
18
19 # Anzeigen der sortierten Daten (optional)
20 print("\nSortierte Daten nach 'Straftaten_insgesamt':")
21 print(df_sortiert.head())
22
23 # Optional: Speichern des sortierten DataFrames in einer neuen Excel-Datei
24 df_sortiert.to_excel('Fallzahlen_sortiert.xlsx', index=False)
25
26 print("\nDie sortierten Daten wurden in 'Fallzahlen_sortiert.xlsx'
    gespeichert.")

```

Listing 3: Skript Testfall 1 Prompt 1 Ausführung 2

```

1 import pandas as pd
2
3 # Pfad zur Excel-Datei
4 excel_datei = 'Fallzahlen.xlsx'
5
6 # Name des Sheets
7 sheet_name = 'Fallzahlen_2023'
8
9 try:
10     # Einlesen der Excel-Datei
11     df = pd.read_excel(excel_datei, sheet_name=sheet_name)
12     print("Excel-Datei erfolgreich eingelesen.")
13
14     # Überprüfen, ob die erforderte Spalte existiert
15     if 'Straftaten_insgesamt' not in df.columns:
16         raise ValueError("Die Spalte 'Straftaten_insgesamt' wurde im
            DataFrame nicht gefunden.")
17
18     # Sortieren des DataFrames nach 'Straftaten_insgesamt' in absteigender
        Reihenfolge
19     df_sorted = df.sort_values(by='Straftaten_insgesamt', ascending=False).
        reset_index(drop=True)
20     print("DataFrame erfolgreich nach 'Straftaten_insgesamt' sortiert.")
21
22     # Optional: Anzeige der sortierten Daten
23     print(df_sorted)
24

```

```

25     # Optional: Speichern des sortierten DataFrames in eine neue Excel-Datei
26     # df_sorted.to_excel('Fallzahlen_sorted.xlsx', index=False)
27     # print("Sortierte Daten wurden in 'Fallzahlen_sorted.xlsx' gespeichert
    .")
28
29 except FileNotFoundError:
30     print(f"Die Datei '{excel_datei}' wurde nicht gefunden.")
31 except ValueError as ve:
32     print(f"Fehler: {ve}")
33 except Exception as e:
34     print(f"Ein unerwarteter Fehler ist aufgetreten: {e}")

```

Listing 4: Skript Testfall 1 Prompt 1 Ausführung 3

```

1 import pandas as pd
2
3 # Pfad zur Excel-Datei
4 excel_datei = 'Fallzahlen.xlsx'
5
6 # Name des Sheets
7 sheet_name = 'Fallzahlen_2023'
8
9 # Excel-Datei einlesen
10 df = pd.read_excel(excel_datei, sheet_name=sheet_name)
11
12 # Daten nach 'Straftaten_insgesamt' in absteigender Reihenfolge sortieren
13 df_sortiert = df.sort_values(by='Straftaten_insgesamt', ascending=False)
14
15 # Optional: Den sortierten DataFrame anzeigen
16 print(df_sortiert)
17
18 # Optional: Den sortierten DataFrame in eine neue Excel-Datei speichern
19 df_sortiert.to_excel('Fallzahlen_sortiert.xlsx', index=False)

```

Listing 5: Skript Testfall 1 Prompt 1 Ausführung 4

```

1 import pandas as pd
2
3 # Pfad zur Excel-Datei
4 dateipfad = 'Fallzahlen.xlsx'
5
6 # Name des Sheets
7 sheet_name = 'Fallzahlen_2023'
8
9 # Einlesen der Excel-Datei
10 try:
11     df = pd.read_excel(dateipfad, sheet_name=sheet_name)
12     print("Daten erfolgreich eingelesen.")
13 except FileNotFoundError:
14     print(f"Die Datei {dateipfad} wurde nicht gefunden.")
15     exit()
16 except ValueError:
17     print(f"Das Sheet '{sheet_name}' existiert nicht in der Datei.")

```

```

18     exit()
19
20 # Überprüfen der Spaltennamen (optional)
21 print("Verfügbare Spalten:")
22 print(df.columns.tolist())
23
24 # Sortieren nach 'Straftaten_insgesamt' in absteigender Reihenfolge
25 # Falls aufsteigend gewünscht ist, setze ascending=True
26 try:
27     df_sortiert = df.sort_values(by='Straftaten_insgesamt', ascending=False)
28     print("Daten erfolgreich sortiert.")
29 except KeyError:
30     print("Die Spalte 'Straftaten_insgesamt' wurde nicht gefunden.")
31     exit()
32
33 # Zurücksetzen des Indexes (optional)
34 df_sortiert.reset_index(drop=True, inplace=True)
35
36 # Anzeigen des sortierten DataFrames
37 print("Sortiertes DataFrame:")
38 print(df_sortiert)

```

Listing 6: Skript Testfall 1 Prompt 1 Ausführung 5

```

1 import pandas as pd
2
3 # Pfad zur Excel-Datei
4 excel_datei = 'Fallzahlen.xlsx'
5
6 # Name des Sheets
7 sheet_name = 'Fallzahlen_2023'
8
9 # LOR-Schlüssel, die ausgeschlossen werden sollen
10 ausschluss_schluessel = [999900, 999999]
11
12 try:
13     # Excel-Datei einlesen
14     df = pd.read_excel(excel_datei, sheet_name=sheet_name)
15
16     # Anzeige der ersten paar Zeilen zur Überprüfung (optional)
17     print("Originaldaten:")
18     print(df.head())
19
20     # Filter anwenden, um die auszuschließenden LOR-Schlüssel zu entfernen
21     df_filtered = df[~df['LOR-Schlüssel'].isin(ausschluss_schluessel)]
22
23     # Anzeige der gefilterten Daten zur Überprüfung (optional)
24     print("\nGefilterte Daten:")
25     print(df_filtered.head())
26
27     # Sortieren nach 'Straftaten_insgesamt' absteigend (größte zuerst)
28     df_sorted = df_filtered.sort_values(by='Straftaten_insgesamt', ascending=
False)

```

```

29
30     # Ergebnis anzeigen (optional)
31     print("\nSortierte Daten:")
32     print(df_sorted)
33
34     # Optional: Das sortierte DataFrame speichern
35     # df_sorted.to_excel('Fallzahlen_sortiert.xlsx', index=False)
36
37 except FileNotFoundError:
38     print(f"Die Datei {excel_datei} wurde nicht gefunden.")
39 except Exception as e:
40     print(f"Ein Fehler ist aufgetreten: {e}")

```

Listing 7: Skript Testfall 1 Prompt 2 Ausführung 1

```

1 import pandas as pd
2
3 # Pfad zur Excel-Datei
4 excel_datei = 'Fallzahlen.xlsx'
5
6 # Name des Sheets
7 sheet_name = 'Fallzahlen_2023'
8
9 # LOR-Schlüssel, die ausgeschlossen werden sollen
10 ausgeschlossene_lor_schluesel = [999900, 999999]
11
12 # Einlesen der Excel-Datei
13 try:
14     df = pd.read_excel(excel_datei, sheet_name=sheet_name, engine='openpyxl')
15     print("Excel-Datei erfolgreich eingelesen.")
16 except FileNotFoundError:
17     print(f"Die Datei {excel_datei} wurde nicht gefunden.")
18     exit(1)
19 except Exception as e:
20     print(f"Ein Fehler ist aufgetreten beim Einlesen der Excel-Datei: {e}")
21     exit(1)
22
23 # Filtern der Zeilen, die nicht zu Bezirken gehören
24 df_filtered = df[~df['LOR-Schlüssel'].isin(ausgeschlossene_lor_schluesel)]
25
26 # Sortieren nach 'Straftaten_insgesamt' in absteigender Reihenfolge
27 df_sorted = df_filtered.sort_values(by='Straftaten_insgesamt', ascending=
    False).reset_index(drop=True)
28
29 # Anzeige des sortierten DataFrames
30 print("Sortierte Daten nach 'Straftaten_insgesamt':")
31 print(df_sorted)
32
33 # Optional: Speichern des sortierten DataFrames in eine neue Excel-Datei
34 # df_sorted.to_excel('Fallzahlen_sortiert.xlsx', index=False)

```

Listing 8: Skript Testfall 1 Prompt 2 Ausführung 2

```

1 import pandas as pd
2
3 # Pfad zur Excel-Datei
4 dateipfad = 'Fallzahlen.xlsx'
5
6 # Name des Sheets
7 sheet_name = 'Fallzahlen_2023'
8
9 # Einlesen der Excel-Datei
10 df = pd.read_excel(dateipfad, sheet_name=sheet_name)
11
12 # Anzeigen der ersten Zeilen zur Überprüfung (optional)
13 #print(df.head())
14
15 # Filtere die Bezirke heraus, indem die unerwünschten LOR-Schlüssel
    ausgeschlossen werden
16 ausgeschlossene_lor = [999900, 999999]
17 bezirke_df = df[~df['LOR-Schlüssel'].isin(ausgeschlossene_lor)]
18
19 # Sortiere die Bezirke nach 'Straftaten_insgesamt' in absteigender
    Reihenfolge
20 bezirke_sortiert = bezirke_df.sort_values(by='Straftaten_insgesamt',
    ascending=False)
21
22 # Optional: Zurücksetzen des Indexes nach dem Sortieren
23 bezirke_sortiert.reset_index(drop=True, inplace=True)
24
25 # Anzeigen des sortierten DataFrames (optional)
26 print(bezirke_sortiert)
27
28 # Wenn Sie den sortierten DataFrame weiter verwenden möchten, steht er als '
    bezirke_sortiert' zur Verfügung

```

Listing 9: Skript Testfall 1 Prompt 2 Ausführung 3

```

1 import pandas as pd
2
3 # Pfad zur Excel-Datei
4 excel_datei = 'Fallzahlen.xlsx'
5
6 # Namen des Sheets
7 sheet_name = 'Fallzahlen_2023'
8
9 # LOR-Schlüssel der Zeilen, die ausgeschlossen werden sollen
10 ausschluss_lor_schluessel = [999900, 999999]
11
12 # Excel-Datei einlesen
13 df = pd.read_excel(excel_datei, sheet_name=sheet_name)
14
15 # Anzeigen der ursprünglichen Daten (optional)
16 print("Ursprüngliche Daten:")
17 print(df)

```

```

18
19 # Filtern der Bezirke, indem die ausgeschlossenen LOR-Schlüssel entfernt
    werden
20 df_gefiltert = df[~df['LOR-Schlüssel'].isin(ausschluss_lor_schluessel)]
21
22 # Anzeigen der gefilterten Daten (optional)
23 print("\nGefilterte Daten (ohne nicht zuzuordnende Bezirke):")
24 print(df_gefiltert)
25
26 # Sortieren nach 'Straftaten_insgesamt' in absteigender Reihenfolge
27 df_sortiert = df_gefiltert.sort_values(by='Straftaten_insgesamt', ascending=
    False)
28
29 # Zurücksetzen des Index (optional, für sauberere Anzeige)
30 df_sortiert.reset_index(drop=True, inplace=True)
31
32 # Anzeigen der sortierten Daten (optional)
33 print("\nSortierte Daten nach 'Straftaten_insgesamt':")
34 print(df_sortiert)
35
36 # Optional: Speichern des sortierten DataFrames in eine neue Excel-Datei
37 # df_sortiert.to_excel('Sortierte_Fallzahlen.xlsx', index=False)

```

Listing 10: Skript Testfall 1 Prompt 2 Ausführung 4

```

1 import pandas as pd
2
3 # Pfad zur Excel-Datei
4 datei_pfad = 'Fallzahlen.xlsx'
5
6 # Namen des Sheets
7 sheet_name = 'Fallzahlen_2023'
8
9 # Einlesen der Excel-Datei
10 try:
11     df = pd.read_excel(datei_pfad, sheet_name=sheet_name)
12     print("Excel-Datei erfolgreich eingelesen.")
13 except FileNotFoundError:
14     print(f"Die Datei {datei_pfad} wurde nicht gefunden.")
15     exit(1)
16 except Exception as e:
17     print(f"Ein Fehler ist aufgetreten beim Einlesen der Datei: {e}")
18     exit(1)
19
20 # Anzeigen der ersten paar Zeilen, um sicherzustellen, dass die Daten korrekt
    eingelesen wurden
21 print("\nErste 5 Zeilen der Originaldaten:")
22 print(df.head())
23
24 # Filterkriterien: Ausschließen der LOR-Schlüssel 999900 und 999999
25 ausschluss_schluessel = [999900, 999999]
26 df_filtered = df[~df['LOR-Schlüssel'].isin(ausschluss_schluessel)]
27

```

```

28 print(f"\nAnzahl der Zeilen nach dem Ausschluss: {len(df_filtered)}")
29
30 # Sortieren nach 'Straftaten_insgesamt' in absteigender Reihenfolge
31 df_sorted = df_filtered.sort_values(by='Straftaten_insgesamt', ascending=
    False).reset_index(drop=True)
32
33 print("\nErste 5 Zeilen der sortierten Daten:")
34 print(df_sorted.head())
35
36 # Optional: Speichern des sortierten DataFrames in eine neue Excel-Datei
37 ausgabe_pfad = 'Fallzahlen_sortiert.xlsx'
38 try:
39     df_sorted.to_excel(ausgabe_pfad, index=False)
40     print(f"\nSortierte Daten wurden erfolgreich in '{ausgabe_pfad}'
    gespeichert.")
41 except Exception as e:
42     print(f"Ein Fehler ist aufgetreten beim Speichern der Datei: {e}")

```

Listing 11: Skript Testfall 1 Prompt 2 Ausführung 5

```

1 import pandas as pd
2 import sys
3
4 def read_excel_file(file_path, sheet_name):
5     """
6     Liest eine Excel-Datei und gibt das entsprechende Sheet als DataFrame zur
7     ück.
8     """
9     try:
10         df = pd.read_excel(file_path, sheet_name=sheet_name)
11         print(f"Erfolgreich das Blatt '{sheet_name}' aus '{file_path}'
        gelesen.")
12         return df
13     except FileNotFoundError:
14         print(f"Fehler: Die Datei '{file_path}' wurde nicht gefunden.")
15         sys.exit(1)
16     except ValueError:
17         print(f"Fehler: Das Blatt '{sheet_name}' existiert nicht in '{
        file_path}'")
18         sys.exit(1)
19     except Exception as e:
20         print(f"Ein unerwarteter Fehler ist aufgetreten: {e}")
21         sys.exit(1)
22
23 def validate_columns(df, required_columns):
24     """
25     Überprüft, ob die erforderlichen Spalten im DataFrame vorhanden sind.
26     """
27     missing_columns = [col for col in required_columns if col not in df.
        columns]
28     if missing_columns:
29         print(f"Fehler: Die folgenden erforderlichen Spalten fehlen im
        DataFrame: {missing_columns}")

```

```

29         sys.exit(1)
30     else:
31         print("Alle erforderlichen Spalten sind vorhanden.")
32
33 def sort_dataframe(df, sort_column, exclude_keys, key_column):
34     """
35     Sortiert den DataFrame nach 'sort_column' absteigend, wobei die Zeilen
36     mit 'exclude_keys'
37     am Ende verbleiben.
38     """
39     # Trennen der Zeilen, die ausgeschlossen werden sollen
40     df_excluded = df[df[key_column].isin(exclude_keys)]
41     df_to_sort = df[~df[key_column].isin(exclude_keys)]
42
43     # Sortieren der relevanten Zeilen
44     df_sorted = df_to_sort.sort_values(by=sort_column, ascending=False)
45
46     # Zusammenführen der sortierten und ausgeschlossenen Zeilen
47     df_final = pd.concat([df_sorted, df_excluded], ignore_index=True)
48
49     print("DataFrame erfolgreich sortiert.")
50     return df_final
51
52 def main():
53     # Parameter
54     file_path = 'Fallzahlen.xlsx'
55     sheet_name = 'Fallzahlen_2023'
56     sort_column = 'Straftaten_insgesamt'
57     key_column = 'LOR-Schlüssel'
58     exclude_keys = [9999900, 999999]
59     output_file = 'Fallzahlen_sortiert.xlsx'
60
61     # Schritt 1: Daten einlesen
62     df = read_excel_file(file_path, sheet_name)
63
64     # Schritt 2: Validierung der erforderlichen Spalten
65     required_columns = [sort_column, key_column]
66     validate_columns(df, required_columns)
67
68     # Schritt 3: Sortierung durchführen
69     df_sorted = sort_dataframe(df, sort_column, exclude_keys, key_column)
70
71     # Optional: Weitere Überprüfung, ob alle Zeilen enthalten sind
72     if len(df_sorted) != len(df):
73         print("Warnung: Die Anzahl der Zeilen nach der Sortierung stimmt
74         nicht mit der ursprünglichen Anzahl überein.")
75
76     # Schritt 4: Ergebnis speichern
77     try:
78         df_sorted.to_excel(output_file, index=False)
79         print(f"Das sortierte DataFrame wurde erfolgreich in '{output_file}'
80         gespeichert.")

```



```

78     except Exception as e:
79         print(f"Fehler beim Speichern der Datei: {e}")
80         sys.exit(1)
81
82 if __name__ == "__main__":
83     main()

```

Listing 12: Skript Testfall 1 Prompt 3 Ausführung 1

```

1  import pandas as pd
2
3  def sort_fallzahlen_excel(file_path: str, sheet_name: str) -> pd.DataFrame:
4      """
5      Liest die Excel-Datei, sortiert die Daten nach 'Straftaten_insgesamt'
6      absteigend,
7      wobei die Zeilen mit den LOR-Schlüsseln 999900 und 999999 am Ende bleiben
8      .
9
10     Parameters:
11     - file_path: Pfad zur Excel-Datei.
12     - sheet_name: Name des Sheets in der Excel-Datei.
13
14     Returns:
15     - Sortierter Pandas DataFrame.
16     """
17     try:
18         # Schritt 1: Einlesen der Excel-Datei
19         df = pd.read_excel(file_path, sheet_name=sheet_name)
20         print(f"Excel-Datei '{file_path}' erfolgreich eingelesen.")
21
22         # Überprüfen, ob die notwendigen Spalten vorhanden sind
23         benötigte_spalten = ['LOR-Schlüssel', 'Straftaten_insgesamt']
24         fehlende_spalten = [spalte for spalte in benötigte_spalten if spalte
25                             not in df.columns]
26         if fehlende_spalten:
27             raise ValueError(f"Fehlende Spalten in der Excel-Datei: {
28                             fehlende_spalten}")
29
30         # Schritt 2: Trennen der Bezirke und Nicht-Bezirke
31         nicht_bezirke_keys = [999900, 999999]
32
33         # Sicherstellen, dass 'LOR-Schlüssel' als numerisch betrachtet wird
34         df['LOR-Schlüssel'] = pd.to_numeric(df['LOR-Schlüssel'], errors='
35         coerce')
36
37         bezirke_df = df[~df['LOR-Schlüssel'].isin(nicht_bezirke_keys)].copy()
38         nicht_bezirke_df = df[df['LOR-Schlüssel'].isin(nicht_bezirke_keys)].
39         copy()
40
41         # Schritt 3: Sortieren der Bezirke nach 'Straftaten_insgesamt'
42         absteigend
43         bezirke_df.sort_values(by='Straftaten_insgesamt', ascending=False,
44                               inplace=True)

```

```

37
38     # Schritt 4: Zusammenfügen der sortierten Bezirke mit den Nicht-
        Bezirken
39     sortiertes_df = pd.concat([bezirke_df, nicht_bezirke_df],
        ignore_index=True)
40
41     print("Daten erfolgreich sortiert.")
42     return sortiertes_df
43
44     except FileNotFoundError:
45         print(f"Datei '{file_path}' wurde nicht gefunden.")
46     except ValueError as ve:
47         print(f"Wertfehler: {ve}")
48     except Exception as e:
49         print(f"Ein unerwarteter Fehler ist aufgetreten: {e}")
50
51 # Anwendung des Skripts
52 if __name__ == "__main__":
53     datei_pfad = 'Fallzahlen.xlsx'
54     sheet = 'Fallzahlen_2023'
55
56     sortiertes_dataframe = sort_fallzahlen_excel(datei_pfad, sheet)
57
58     if sortiertes_dataframe is not None:
59         # Schritt 5: Optional - Speichern des sortierten DataFrames in einer
        neuen Excel-Datei
60         ausgabe_pfad = 'Sortierte_Fallzahlen.xlsx'
61         try:
62             sortiertes_dataframe.to_excel(ausgabe_pfad, index=False)
63             print(f"Sortiertes DataFrame wurde erfolgreich in '{ausgabe_pfad
        }' gespeichert.")
64         except Exception as e:
65             print(f"Fehler beim Speichern der Datei: {e}")

```

Listing 13: Skript Testfall 1 Prompt 3 Ausführung 2

```

1 import pandas as pd
2 import sys
3
4 def main():
5     excel_file = 'Fallzahlen.xlsx'
6     sheet_name = 'Fallzahlen_2023'
7     lor_keys_to_exclude = [999900, 999999]
8
9     try:
10         # Schritt 1: Einlesen der Excel-Datei
11         df = pd.read_excel(excel_file, sheet_name=sheet_name)
12         print("Excel-Datei erfolgreich eingelesen.")
13     except FileNotFoundError:
14         print(f"Fehler: Die Datei '{excel_file}' wurde nicht gefunden.")
15         sys.exit(1)
16     except ValueError as e:
17         print(f"Fehler beim Einlesen des Sheets '{sheet_name}': {e}")

```

```

18     sys.exit(1)
19 except Exception as e:
20     print(f"Ein unerwarteter Fehler ist aufgetreten: {e}")
21     sys.exit(1)
22
23 # Überprüfen, ob die notwendigen Spalten vorhanden sind
24 required_columns = ['LOR-Schlüssel', 'Straftaten_insgesamt']
25 missing_columns = [col for col in required_columns if col not in df.
26 columns]
27 if missing_columns:
28     print(f"Fehler: Fehlende Spalten in den Daten: {missing_columns}")
29     sys.exit(1)
30
31 # Sicherstellen, dass 'LOR-Schlüssel' numerisch ist
32 try:
33     df['LOR-Schlüssel'] = pd.to_numeric(df['LOR-Schlüssel'])
34 except ValueError:
35     print("Fehler: Die Spalte 'LOR-Schlüssel' enthält nicht-numerische
36 Werte.")
37     sys.exit(1)
38
39 # Schritt 2: Trennen der Bezirke und der auszuschließenden Schlüssel
40 df_bezirke = df[~df['LOR-Schlüssel'].isin(lor_keys_to_exclude)]
41 df_excluded = df[df['LOR-Schlüssel'].isin(lor_keys_to_exclude)]
42
43 # Schritt 3: Sortieren der Bezirke nach 'Straftaten_insgesamt' absteigend
44 if 'Straftaten_insgesamt' not in df_bezirke.columns:
45     print("Fehler: Die Spalte 'Straftaten_insgesamt' ist nicht in den
46 Daten vorhanden.")
47     sys.exit(1)
48
49 try:
50     df_bezirke_sorted = df_bezirke.sort_values(by='Straftaten_insgesamt',
51 ascending=False)
52 except Exception as e:
53     print(f"Fehler beim Sortieren der Daten: {e}")
54     sys.exit(1)
55
56 # Schritt 4: Zusammenführen der sortierten Bezirke mit den
57 ausgeschlossenen Zeilen
58 df_final = pd.concat([df_bezirke_sorted, df_excluded], ignore_index=True)
59
60 # Optional: Ausgabe oder Speicherung des finalen DataFrames
61 # Beispiel: Anzeigen der ersten 5 Zeilen
62 print("Sortiertes DataFrame:")
63 print(df_final.head())
64
65 # Beispiel: Speichern in eine neue Excel-Datei
66 output_file = 'Fallzahlen_sortiert.xlsx'
67 try:
68     df_final.to_excel(output_file, index=False)
69     print(f"Das sortierte DataFrame wurde erfolgreich in '{output_file}'

```

```

    gespeichert.")
65     except Exception as e:
66         print(f"Fehler beim Speichern der Datei '{output_file}': {e}")
67         sys.exit(1)
68
69 if __name__ == "__main__":
70     main()

```

Listing 14: Skript Testfall 1 Prompt 3 Ausführung 3

```

1  import pandas as pd
2
3  def sort_fallzahlen_excel(file_path: str, sheet_name: str) -> pd.DataFrame:
4      """
5      Reads the specified sheet from an Excel file, sorts the data by '
6      Straftaten_insgesamt'
7      in descending order excluding rows with LOR-Schlüssel 999900 and 999999,
8      which are
9      appended at the end.
10
11     Parameters:
12     - file_path: Path to the Excel file.
13     - sheet_name: Name of the sheet to read.
14
15     Returns:
16     - A sorted pandas DataFrame.
17     """
18     try:
19         # Read the Excel sheet
20         df = pd.read_excel(file_path, sheet_name=sheet_name)
21         print(f"Successfully read the sheet '{sheet_name}' from '{file_path}'")
22     except FileNotFoundError:
23         print(f"Error: The file '{file_path}' was not found.")
24         return pd.DataFrame()
25     except ValueError:
26         print(f"Error: The sheet '{sheet_name}' does not exist in '{file_path}'")
27         return pd.DataFrame()
28     except Exception as e:
29         print(f"An unexpected error occurred while reading the Excel file: {e}")
30         return pd.DataFrame()
31
32     # Check if necessary columns exist
33     required_columns = ['LOR-Schlüssel', 'Straftaten_insgesamt']
34     missing_columns = [col for col in required_columns if col not in df.columns]
35     if missing_columns:
36         print(f"Error: The following required columns are missing from the data: {missing_columns}")
37         return pd.DataFrame()

```

```

37 # Separate the rows with LOR-Schlüssel 999900 and 999999
38 exclusion_keys = [999900, 999999]
39 excluded_df = df[df['LOR-Schlüssel'].isin(exclusion_keys)]
40 included_df = df[~df['LOR-Schlüssel'].isin(exclusion_keys)]
41
42 if excluded_df.empty:
43     print("Warning: No rows found with LOR-Schlüssel 999900 or 999999.")
44
45 # Sort the included DataFrame by 'Straftaten_insgesamt' in descending
46 # order
47 try:
48     included_sorted_df = included_df.sort_values(by='Straftaten_insgesamt',
49     ascending=False)
50     print("Successfully sorted the included data by 'Straftaten_insgesamt'
51     in descending order.")
52 except KeyError:
53     print("Error: The column 'Straftaten_insgesamt' does not exist in the
54     data.")
55     return pd.DataFrame()
56 except Exception as e:
57     print(f"An unexpected error occurred during sorting: {e}")
58     return pd.DataFrame()
59
60 # Concatenate the sorted included data with the excluded data
61 sorted_df = pd.concat([included_sorted_df, excluded_df], ignore_index=
62 True)
63 print("Successfully concatenated the excluded rows at the end of the
64 sorted data.")
65
66 return sorted_df
67
68 if __name__ == "__main__":
69     # Define the Excel file path and sheet name
70     excel_file = 'Fallzahlen.xlsx'
71     sheet = 'Fallzahlen_2023'
72
73     # Get the sorted DataFrame
74     sorted_fallzahlen_df = sort_fallzahlen_excel(excel_file, sheet)
75
76     # Check if the DataFrame is not empty before proceeding
77     if not sorted_fallzahlen_df.empty:
78         print("Sorted DataFrame:")
79         print(sorted_fallzahlen_df)
80         # Optionally, you can save the sorted DataFrame to a new Excel file
81         # or another format
82         # sorted_fallzahlen_df.to_excel('Sorted_Fallzahlen.xlsx', index=False
83         # )
84     else:
85         print("No data to display due to previous errors.")

```

Listing 15: Skript Testfall 1 Prompt 3 Ausführung 4

```

1 import pandas as pd

```

```

2
3 def sort_fallzahlen_excel(file_path: str, sheet_name: str) -> pd.DataFrame:
4     """
5     Liest die Daten aus dem angegebenen Excel-Sheet, sortiert sie nach '
6     Straftaten_insgesamt' absteigend,
7     wobei die Zeilen mit den LOR-Schlüsseln 999900 und 999999 am Ende des
8     DataFrames stehen.
9
10    Parameters:
11    - file_path: Pfad zur Excel-Datei.
12    - sheet_name: Name des Sheets, das eingelesen werden soll.
13
14    Returns:
15    - Ein sortierter Pandas DataFrame.
16    """
17    try:
18        # Schritt 1: Einlesen der Excel-Datei
19        df = pd.read_excel(file_path, sheet_name=sheet_name)
20        print(f"Erfolgreich das Sheet '{sheet_name}' aus '{file_path}'
21        eingelesen.")
22    except FileNotFoundError:
23        raise FileNotFoundError(f"Die Datei '{file_path}' wurde nicht
24        gefunden.")
25    except ValueError:
26        raise ValueError(f"Das Sheet '{sheet_name}' existiert nicht in '{
27        file_path}'")
28    except Exception as e:
29        raise Exception(f"Ein unerwarteter Fehler ist beim Einlesen der Excel
30        -Datei aufgetreten: {e}")
31
32    # Überprüfen, ob die notwendigen Spalten vorhanden sind
33    required_columns = ['LOR-Schlüssel', 'Straftaten_insgesamt']
34    missing_columns = [col for col in required_columns if col not in df.
35    columns]
36    if missing_columns:
37        raise KeyError(f"Die folgenden erforderlichen Spalten fehlen im
38        DataFrame: {missing_columns}")
39    else:
40        print("Alle erforderlichen Spalten sind vorhanden.")
41
42    # Schritt 2: Separiere die Bezirke und die nicht zuzuordnenden Einträge
43    # Definiere die LOR-Schlüssel, die nicht sortiert werden sollen
44    exclude_lor_keys = [999900, 999999]
45
46    # Konvertiere 'LOR-Schlüssel' zu integer, falls nötig
47    if df['LOR-Schlüssel'].dtype != 'int64' and df['LOR-Schlüssel'].dtype !=
48    'int32':
49        try:
50            df['LOR-Schlüssel'] = df['LOR-Schlüssel'].astype(int)
51            print("'LOR-Schlüssel' erfolgreich in Integer konvertiert.")
52        except ValueError:
53            raise ValueError("'LOR-Schlüssel' enthält nicht-numerische Werte,

```

```

    die nicht in Integer konvertiert werden können.")
45
46 # Teile den DataFrame in Bezirke und Nicht-Bezirke
47 bezirke_df = df[~df['LOR-Schlüssel'].isin(exclude_lor_keys)]
48 nicht_bezirke_df = df[df['LOR-Schlüssel'].isin(exclude_lor_keys)]
49
50 print(f"Anzahl der Bezirke: {bezirke_df.shape[0]}")
51 print(f"Anzahl der nicht zuzuordnenden Einträge: {nicht_bezirke_df.shape
[0]}")
52
53 # Schritt 3: Sortiere die Bezirke nach 'Straftaten_insgesamt' absteigend
54 try:
55     bezirke_sorted = bezirke_df.sort_values(by='Straftaten_insgesamt',
ascending=False)
56     print("Bezirke erfolgreich nach 'Straftaten_insgesamt' absteigend
sortiert.")
57 except KeyError:
58     raise KeyError("Die Spalte 'Straftaten_insgesamt' wurde nicht
gefunden und kann nicht zum Sortieren verwendet werden.")
59 except Exception as e:
60     raise Exception(f"Ein unerwarteter Fehler ist beim Sortieren der
Bezirke aufgetreten: {e}")
61
62 # Schritt 4: Konkateneriere die sortierten Bezirke mit den nicht
zuzuordnenden Einträgen
63 final_df = pd.concat([bezirke_sorted, nicht_bezirke_df], ignore_index=
True)
64 print("Bezirke und nicht zuzuordnende Einträge erfolgreich kombiniert.")
65
66 return final_df
67
68 if __name__ == "__main__":
69     # Definiere den Pfad zur Excel-Datei und den Sheet-Namen
70     excel_file = 'Fallzahlen.xlsx'
71     sheet = 'Fallzahlen_2023'
72
73     try:
74         # Führe die Sortierung durch und hole das sortierte DataFrame
75         sortiertes_df = sort_fallzahlen_excel(excel_file, sheet)
76         print("Daten erfolgreich sortiert.")
77
78         # Optional: Anzeige der ersten paar Zeilen des sortierten DataFrames
79         print(sortiertes_df.head())
80
81         # Optional: Speichern des sortierten DataFrames in eine neue Excel-
Datei
82         # sortiertes_df.to_excel('Fallzahlen_sortiert.xlsx', index=False)
83         # print("Sortiertes DataFrame wurde in 'Fallzahlen_sortiert.xlsx'
gespeichert.")
84
85     except Exception as e:
86         print(f"Fehler: {e}")

```

## 8.2.2 Testfall 2

```

1 import pandas as pd
2
3 def lese_und_sortiere_fallzahlen(excel_datei):
4     """
5     Liest alle Sheets aus der angegebenen Excel-Datei, kombiniert die Daten,
6     sortiert sie nach 'Straftaten_insgesamt' in absteigender Reihenfolge
7     und gibt den sortierten DataFrame zurück.
8
9     :param excel_datei: Pfad zur Excel-Datei (z.B. 'Fallzahlen.xlsx')
10    :return: Sortierter Pandas DataFrame
11    """
12    try:
13        # Alle Sheets der Excel-Datei lesen. sheet_name=None liest alle
14        # Sheets als Dict.
15        alle_sheets = pd.read_excel(excel_datei, sheet_name=None)
16
17        # Liste zur Speicherung aller DataFrames
18        dataframe_liste = []
19
20        # Durch alle Sheets iterieren und die DataFrames zur Liste hinzufügen
21        for sheet_name, df in alle_sheets.items():
22            # Optional: Hinzufügen einer Spalte mit dem Sheet-Namen, falls nö
23            # tig
24            # df['Sheet_Name'] = sheet_name
25            dataframe_liste.append(df)
26
27        # Alle DataFrames zu einem einzigen DataFrame kombinieren
28        kombiniertes_df = pd.concat(dataframe_liste, ignore_index=True)
29
30        # Prüfen, ob die Spalte 'Straftaten_insgesamt' existiert
31        if 'Straftaten_insgesamt' not in kombiniertes_df.columns:
32            raise ValueError("Die Spalte 'Straftaten_insgesamt' wurde in den
33            Daten nicht gefunden.")
34
35        # Sortieren nach 'Straftaten_insgesamt' in absteigender Reihenfolge
36        sortiertes_df = kombiniertes_df.sort_values(by='Straftaten_insgesamt',
37            ascending=False).reset_index(drop=True)
38
39        return sortiertes_df
40
41    except FileNotFoundError:
42        print(f"Die Datei {excel_datei} wurde nicht gefunden.")
43    except Exception as e:
44        print(f"Ein Fehler ist aufgetreten: {e}")
45
46 if __name__ == "__main__":

```



```

43     # Pfad zur Excel-Datei
44     dateipfad = 'Fallzahlen.xlsx'
45
46     # Funktion aufrufen und sortierten DataFrame erhalten
47     df_sortiert = lese_und_sortiere_fallzahlen(dateipfad)
48
49     if df_sortiert is not None:
50         # Sortierten DataFrame anzeigen
51         print(df_sortiert)
52
53         # Optional: Sortierten DataFrame in eine neue Excel-Datei speichern
54         df_sortiert.to_excel('Fallzahlen_sortiert.xlsx', index=False)

```

Listing 17: Skript Testfall 2 Prompt 1 Ausführung 1

```

1  import pandas as pd
2
3  # Pfad zur Excel-Datei
4  excel_datei = 'Fallzahlen.xlsx'
5
6  # Einlesen aller Sheets in ein Dictionary von DataFrames
7  try:
8      alle_sheets = pd.read_excel(excel_datei, sheet_name=None)
9  except FileNotFoundError:
10     print(f"Die Datei {excel_datei} wurde nicht gefunden.")
11     exit(1)
12 except Exception as e:
13     print(f"Fehler beim Einlesen der Excel-Datei: {e}")
14     exit(1)
15
16 # Liste zur Speicherung der relevanten Daten aus allen Sheets
17 daten_liste = []
18
19 # Durch alle DataFrames iterieren und relevante Daten extrahieren
20 for sheet_name, df in alle_sheets.items():
21     # Überprüfen, ob die notwendigen Spalten vorhanden sind
22     if 'Bezirke' in df.columns and 'Straftaten_insgesamt' in df.columns:
23         # Auswahl der relevanten Spalten
24         df_relevant = df[['Bezirke', 'Straftaten_insgesamt']]
25         daten_liste.append(df_relevant)
26     else:
27         print(f"Sheet '{sheet_name}' enthält nicht die erforderlichen Spalten")
28
29 # Kombinieren aller relevanten Daten
30 if daten_liste:
31     alle_daten = pd.concat(daten_liste)
32 else:
33     print("Keine relevanten Daten gefunden.")
34     exit(1)
35
36 # Aggregation der Straftaten pro Bezirk
37 aggregierte_daten = alle_daten.groupby('Bezirke', as_index=False)['

```

```

    Straftaten_insgesamt'].sum()
38
39 # Sortieren der Bezirke nach der Anzahl der Straftaten (absteigend)
40 aggregierte_daten_sortiert = aggregierte_daten.sort_values(by='
    Straftaten_insgesamt', ascending=False).reset_index(drop=True)
41
42 # Optional: Anzeige des resultierenden DataFrames
43 print(aggregierte_daten_sortiert)
44
45 # Optional: Speichern des DataFrames in eine neue Excel-Datei
46 # aggregierte_daten_sortiert.to_excel('Aggregierte_Fallzahlen.xlsx', index=
    False)

```

Listing 18: Skript Testfall 2 Prompt 1 Ausführung 2

```

1 import pandas as pd
2
3 # Pfad zur Excel-Datei
4 excel_datei = 'Fallzahlen.xlsx'
5
6 # Alle Sheet-Namen aus der Excel-Datei auslesen
7 xls = pd.ExcelFile(excel_datei)
8 sheet_namen = xls.sheet_names
9
10 # Liste zur Speicherung der DataFrames aller Sheets
11 daten_frames = []
12
13 # Iteriere über jedes Sheet und lese die Daten ein
14 for sheet in sheet_namen:
15     df = pd.read_excel(excel_datei, sheet_name=sheet)
16     daten_frames.append(df)
17
18 # Alle DataFrames zu einem einzigen DataFrame zusammenführen
19 kombiniert_df = pd.concat(daten_frames, ignore_index=True)
20
21 # Optional: Duplikate entfernen, falls Bezirke in mehreren Sheets vorkommen
22 # kombiniert_df = kombiniert_df.drop_duplicates(subset=['Bezirke'])
23
24 # Gruppieren nach 'Bezirke' und Summieren der 'Straftaten_insgesamt'
25 aggregiert_df = kombiniert_df.groupby('Bezirke', as_index=False)['
    Straftaten_insgesamt'].sum()
26
27 # Sortieren nach der Anzahl der Straftaten in absteigender Reihenfolge
28 sortiert_df = aggregiert_df.sort_values(by='Straftaten_insgesamt', ascending=
    False)
29
30 # Optional: Index zurücksetzen
31 sortiert_df = sortiert_df.reset_index(drop=True)
32
33 # Ergebnis anzeigen
34 print(sortiert_df)
35
36 # Optional: Das sortierte Ergebnis in eine neue Excel- oder CSV-Datei

```

```

    speichern
37 # sortiert_df.to_excel('Aggregierte_Fallzahlen.xlsx', index=False)
38 # sortiert_df.to_csv('Aggregierte_Fallzahlen.csv', index=False)

```

Listing 19: Skript Testfall 2 Prompt 1 Ausführung 3

```

1 import pandas as pd
2
3 # Pfad zur Excel-Datei
4 excel_file = 'Fallzahlen.xlsx'
5
6 # Schritt 1: Alle Sheets der Excel-Datei einlesen
7 # Das Argument sheet_name=None liest alle Sheets und gibt ein Dictionary zurück
8 all_sheets = pd.read_excel(excel_file, sheet_name=None)
9
10 # Schritt 2: Daten aus allen Sheets zusammenführen
11 # Wir gehen davon aus, dass alle Sheets die gleiche Struktur haben
12 # Daher können wir sie einfach mit pd.concat zusammenfügen
13 combined_df = pd.concat(all_sheets.values(), ignore_index=True)
14
15 # Schritt 3: Daten nach 'Bezirke' gruppieren und 'Straftaten_insgesamt'
    summieren
16 # Dies aggregiert die Straftaten pro Bezirk über alle Sheets hinweg
17 grouped_df = combined_df.groupby('Bezirke')['Straftaten_insgesamt'].sum().
    reset_index()
18
19 # Schritt 4: Die aggregierten Daten nach 'Straftaten_insgesamt' sortieren (
    absteigend)
20 sorted_df = grouped_df.sort_values(by='Straftaten_insgesamt', ascending=False)
    .reset_index(drop=True)
21
22 # Optional: Anzeigen des sortierten DataFrames
23 print(sorted_df)
24
25 # Optional: Den sortierten DataFrame in eine neue Excel-Datei speichern
26 sorted_df.to_excel('Aggregierte_Straftaten_pro_Bezirk.xlsx', index=False)

```

Listing 20: Skript Testfall 2 Prompt 1 Ausführung 4

```

1 import pandas as pd
2
3 def lese_und_sortiere_fallzahlen(excel_datei):
4     """
5     Liest alle Sheets aus der gegebenen Excel-Datei, kombiniert die Daten,
6     sortiert sie nach der Gesamtanzahl der Straftaten pro Bezirk und
7     gibt das sortierte DataFrame zurück.
8
9     :param excel_datei: Pfad zur Excel-Datei 'Fallzahlen.xlsx'
10    :return: Sortiertes pandas DataFrame
11    """
12    try:
13        # Alle Sheets lesen; sheet_name=None gibt ein Dictionary zurück

```

```

14     alle_sheets = pd.read_excel(excel_datei, sheet_name=None)
15     print(f"Anzahl der gelesenen Sheets: {len(alle_sheets)}")
16 except FileNotFoundError:
17     print(f"Die Datei '{excel_datei}' wurde nicht gefunden.")
18     return None
19 except Exception as e:
20     print(f"Ein Fehler ist aufgetreten: {e}")
21     return None
22
23 # Kombinieren aller DataFrames in einen
24 try:
25     kombiniertes_df = pd.concat(alle_sheets.values(), ignore_index=True)
26     print(f"Anzahl der Zeilen nach dem Kombinieren: {len(kombiniertes_df)}")
27 except ValueError as ve:
28     print(f"Fehler beim Kombinieren der Daten: {ve}")
29     return None
30
31 # Sicherstellen, dass die Spalte 'Straftaten_insgesamt' numerisch ist
32 if not pd.api.types.is_numeric_dtype(kombiniertes_df['Straftaten_insgesamt']):
33     kombiniertes_df['Straftaten_insgesamt'] = pd.to_numeric(
34         kombiniertes_df['Straftaten_insgesamt'], errors='coerce'
35     )
36     kombiniertes_df = kombiniertes_df.dropna(subset=['Straftaten_insgesamt'])
37
38 # Sortieren nach 'Straftaten_insgesamt' absteigend
39 sortiertes_df = kombiniertes_df.sort_values(
40     by='Straftaten_insgesamt', ascending=False
41 ).reset_index(drop=True)
42
43 print("Die Daten wurden erfolgreich sortiert.")
44 return sortiertes_df
45
46 def main():
47     excel_datei = 'Fallzahlen.xlsx'
48     df = lese_und_sortiere_fallzahlen(excel_datei)
49     if df is not None:
50         # Optional: Das sortierte DataFrame anzeigen
51         print(df)
52
53         # Optional: Das sortierte DataFrame speichern
54         # df.to_csv('Sortierte_Fallzahlen.csv', index=False, encoding='utf-8-sig')
55         # print("Das sortierte DataFrame wurde als 'Sortierte_Fallzahlen.csv' gespeichert.")
56
57 if __name__ == "__main__":
58     main()

```

Listing 21: Skript Testfall 2 Prompt 1 Ausführung 5

```

1 import pandas as pd
2
3 def zusammenfuegen_und_sortieren(excel_datei):
4     """
5     Liest alle Sheets einer Excel-Datei, fügt sie zusammen,
6     entfernt bestimmte LOR-Schlüssel, sortiert nach 'Straftaten_insgesamt'
7     und gibt den resultierenden DataFrame zurück.
8
9     Parameters:
10    - excel_datei: str, Pfad zur Excel-Datei.
11
12    Returns:
13    - pd.DataFrame: Gefilterter und sortierter DataFrame.
14    """
15
16    # Laden Sie die Excel-Datei und holen Sie alle Sheet-Namen
17    xls = pd.ExcelFile(excel_datei)
18    sheet_namen = xls.sheet_names
19
20    # Liste zur Speicherung der DataFrames aus jedem Sheet
21    df_list = []
22
23    # Durchlaufen Sie jede Sheet und lesen Sie die Daten
24    for sheet in sheet_namen:
25        df = pd.read_excel(excel_datei, sheet_name=sheet)
26        df_list.append(df)
27        print(f"Gesheet '{sheet}' erfolgreich geladen.")
28
29    # Verbinden Sie alle DataFrames zu einem einzigen DataFrame
30    kombiniert_df = pd.concat(df_list, ignore_index=True)
31    print("Alle Sheets wurden erfolgreich zusammengeführt.")
32
33    # Entfernen Sie die Zeilen mit den LOR-Schlüsseln 999900 und 999999
34    filtered_df = kombiniert_df[~kombiniert_df['LOR-Schlüssel'].isin([999900,
35        999999])]
36    print("Unerwünschte LOR-Schlüssel wurden entfernt.")
37
38    # Sortieren Sie den DataFrame nach 'Straftaten_insgesamt' in absteigender
39    Reihenfolge
40    sorted_df = filtered_df.sort_values(by='Straftaten_insgesamt', ascending=
41        False).reset_index(drop=True)
42    print("DataFrame wurde nach 'Straftaten_insgesamt' sortiert.")
43
44    return sorted_df
45
46 if __name__ == "__main__":
47     # Pfad zur Excel-Datei
48     excel_pfad = 'Fallzahlen.xlsx'
49
50     # Rufen Sie die Funktion auf und speichern Sie das Ergebnis in 'final_df'
51     final_df = zusammenfuegen_und_sortieren(excel_pfad)

```

```

50 # Optional: Zeigen Sie die ersten paar Zeilen des finalen DataFrames an
51 print(final_df.head())
52
53 # Optional: Speichern Sie den finalen DataFrame als neue Excel-Datei
54 # final_df.to_excel('Zusammengefuellte_Fallzahlen.xlsx', index=False)

```

Listing 22: Skript Testfall 2 Prompt 2 Ausführung 1

```

1 import pandas as pd
2
3 def load_and_process_excel(file_path):
4     """
5     Lädt alle Sheets aus der angegebenen Excel-Datei, fügt sie zusammen,
6     filtert unerwünschte Zeilen, sortiert die Daten und gibt einen DataFrame
7     zurück.
8
9     Parameters:
10         file_path (str): Der Pfad zur Excel-Datei.
11
12     Returns:
13         pd.DataFrame: Der verarbeitete DataFrame.
14     """
15     try:
16         # Lade alle Sheets in ein Dictionary von DataFrames
17         all_sheets = pd.read_excel(file_path, sheet_name=None, dtype={'LOR-
18 Schlüssel': str})
19
20         # Liste zum Speichern der DataFrames
21         df_list = []
22
23         for sheet_name, df in all_sheets.items():
24             print(f"Lade Sheet: {sheet_name} mit {len(df)} Zeilen.")
25             df_list.append(df)
26
27         # Kombiniere alle DataFrames in einen einzigen DataFrame
28         combined_df = pd.concat(df_list, ignore_index=True)
29         print(f"Gesamtanzahl der Zeilen nach dem Zusammenführen: {len(
30 combined_df)}")
31
32         # Entferne Zeilen mit LOR-Schlüssel 999900 und 999999
33         filtered_df = combined_df[~combined_df['LOR-Schlüssel'].isin([999900,
34 999999])]
35         print(f"Anzahl der Zeilen nach dem Filtern: {len(filtered_df)}")
36
37         # Sortiere nach 'Straftaten_insgesamt' absteigend
38         sorted_df = filtered_df.sort_values(by='Straftaten_insgesamt',
39 ascending=False)
40         print("Daten wurden nach 'Straftaten_insgesamt' sortiert.")
41
42         # Optional: Setze den Index zurück
43         sorted_df.reset_index(drop=True, inplace=True)
44
45         return sorted_df

```

```

41
42     except FileNotFoundError:
43         print(f"Die Datei {file_path} wurde nicht gefunden.")
44     except Exception as e:
45         print(f"Ein Fehler ist aufgetreten: {e}")
46
47 if __name__ == "__main__":
48     # Pfad zur Excel-Datei
49     excel_file = 'Fallzahlen.xlsx'
50
51     # Verarbeite die Excel-Datei
52     final_df = load_and_process_excel(excel_file)
53
54     if final_df is not None:
55         # Zeige die ersten paar Zeilen des finalen DataFrames
56         print("\nErgebnis:")
57         print(final_df.head())
58
59         # Optional: Speichere den DataFrame in eine neue Excel- oder CSV-
        Datei
60         # final_df.to_excel('Zusammengefuegte_Fallzahlen.xlsx', index=False)
61         # final_df.to_csv('Zusammengefuegte_Fallzahlen.csv', index=False)

```

Listing 23: Skript Testfall 2 Prompt 2 Ausführung 2

```

1 import pandas as pd
2
3 # Pfad zur Excel-Datei
4 excel_file = 'Fallzahlen.xlsx'
5
6 # LOR-Schlüssel, die ausgeschlossen werden sollen
7 exclude_lor_keys = [999900, 999999]
8
9 # Alle Sheets laden
10 sheets_dict = pd.read_excel(excel_file, sheet_name=None)
11
12 # Liste zur Speicherung der bereinigten DataFrames
13 filtered_dfs = []
14
15 for sheet_name, df in sheets_dict.items():
16     # Sicherstellen, dass die benötigten Spalten vorhanden sind
17     if 'LOR-Schlüssel' not in df.columns or 'Bezirke' not in df.columns:
18         print(f"Sheet '{sheet_name}' übersprungen, da erforderliche Spalten
        fehlen.")
19         continue
20
21     # Ausschließen der unerwünschten LOR-Schlüssel
22     df_filtered = df[~df['LOR-Schlüssel'].isin(exclude_lor_keys)].copy()
23
24     # Optional: Hinzufügen einer Spalte zur Kennzeichnung des Sheets
25     df_filtered['Sheet'] = sheet_name
26
27     filtered_dfs.append(df_filtered)

```

```

28
29 # Überprüfen, ob es DataFrames zum Zusammenführen gibt
30 if not filtered_dfs:
31     raise ValueError("Keine gültigen Daten zum Zusammenführen gefunden.")
32
33 # Zusammenführen aller DataFrames auf Basis von 'LOR-Schlüssel' und 'Bezirke'
34 # Falls es gemeinsame Spalten außer den Schlüsseln gibt, werden sie mit
    Suffixen versehen
35 from functools import reduce
36
37 # Füge die DataFrames schrittweise zusammen
38 merged_df = reduce(lambda left, right: pd.merge(left, right, on=['LOR-Schlüssel', 'Bezirke'], how='outer', suffixes=('', '_dup')), filtered_dfs)
39
40 # Optional: Entfernen von doppelten Spalten, die durch die Zusammenführung
    entstehen könnten
41 merged_df = merged_df.loc[:, ~merged_df.columns.duplicated()]
42
43 # Sortieren nach 'Straftaten_insgesamt' absteigend
44 if 'Straftaten_insgesamt' in merged_df.columns:
45     sorted_df = merged_df.sort_values(by='Straftaten_insgesamt', ascending=False)
46 else:
47     print("'Straftaten_insgesamt' Spalte nicht gefunden. Sortierung übersprungen.")
48     sorted_df = merged_df
49
50 # Reset des Indexes
51 sorted_df.reset_index(drop=True, inplace=True)
52
53 # Ergebnis anzeigen
54 print(sorted_df)
55
56 # Optional: Speichern des sortierten DataFrames in eine neue Excel-Datei
57 sorted_df.to_excel('Zusammengefuegte_Fallzahlen.xlsx', index=False)

```

Listing 24: Skript Testfall 2 Prompt 2 Ausführung 3

```

1 import pandas as pd
2
3 def merge_and_sort_fallzahlen(excel_file):
4     """
5     Liest alle Sheets der angegebenen Excel-Datei, filtert unerwünschte LOR-
        Schlüssel,
6     fasst die Daten der Bezirke zusammen, sortiert sie nach
        Straftaten_insgesamt
7     und gibt das resultierende DataFrame zurück.
8
9     Parameters:
10     - excel_file (str): Pfad zur Excel-Datei (z.B. 'Fallzahlen.xlsx')
11
12     Returns:
13     - pd.DataFrame: Gefiltertes und sortiertes DataFrame

```



```

14     """
15
16     # Definiere die unerwünschten LOR-Schlüssel
17     exclude_lor_keys = [999900, 999999]
18
19     try:
20         # Lade alle Sheets in ein Dictionary von DataFrames
21         sheets_dict = pd.read_excel(excel_file, sheet_name=None)
22         print(f"Es wurden {len(sheets_dict)} Sheets gefunden und geladen.")
23     except FileNotFoundError:
24         print(f"Die Datei '{excel_file}' wurde nicht gefunden.")
25         return None
26     except Exception as e:
27         print(f"Ein Fehler ist beim Lesen der Excel-Datei aufgetreten: {e}")
28         return None
29
30     # Liste zur Speicherung gefilterter DataFrames
31     filtered_dfs = []
32
33     # Iteriere über jedes Sheet und filtere die unerwünschten Zeilen
34     for sheet_name, df in sheets_dict.items():
35         print(f"Verarbeite Sheet: {sheet_name}")
36
37         # Überprüfe, ob die erforderlichen Spalten vorhanden sind
38         required_columns = ['LOR-Schlüssel', 'Bezirke', 'Straftaten_insgesamt']
39
40         if not all(col in df.columns for col in required_columns):
41             print(f"Sheet '{sheet_name}' fehlt eine oder mehrere
42             erforderliche Spalten. Überspringe dieses Sheet.")
43             continue
44
45         # Filtere die unerwünschten LOR-Schlüssel
46         df_filtered = df[~df['LOR-Schlüssel'].isin(exclude_lor_keys)].copy()
47         print(f"Anzahl der Bezirke nach Filtern in Sheet '{sheet_name}': {len
48         (df_filtered)}")
49
50         # Optional: Falls du weitere Daten aggregieren möchtest, kannst du
51         hier weitere Schritte hinzufügen
52
53         filtered_dfs.append(df_filtered)
54
55     if not filtered_dfs:
56         print("Keine gültigen Daten zum Zusammenfügen gefunden.")
57         return None
58
59     # Füge alle gefilterten DataFrames zusammen
60     combined_df = pd.concat(filtered_dfs, ignore_index=True)
61     print(f"Gesamtanzahl der Zeilen nach dem Zusammenfügen: {len(combined_df)
62     }")
63
64     # Gruppiere die Daten nach 'LOR-Schlüssel' und 'Bezirke' und summiere '
65     Straftaten_insgesamt'

```

```

60 aggregated_df = combined_df.groupby(['LOR-Schlüssel', 'Bezirke'],
as_index=False)['Straftaten_insgesamt'].sum()
61 print("Daten wurden nach 'LOR-Schlüssel' und 'Bezirke' aggregiert.")
62
63 # Sortiere das DataFrame nach 'Straftaten_insgesamt' in absteigender
Reihenfolge
64 sorted_df = aggregated_df.sort_values(by='Straftaten_insgesamt',
ascending=False).reset_index(drop=True)
65 print("Daten wurden nach 'Straftaten_insgesamt' sortiert.")
66
67 return sorted_df
68
69 if __name__ == "__main__":
70     # Pfad zur Excel-Datei
71     excel_file = 'Fallzahlen.xlsx'
72
73     # Rufe die Funktion auf und erhalte das sortierte DataFrame
74     result_df = merge_and_sort_fallzahlen(excel_file)
75
76     if result_df is not None:
77         # Zeige die ersten paar Zeilen des Ergebnisses an
78         print("\nSortiertes Ergebnis:")
79         print(result_df.head())
80
81         # Optional: Speichere das Ergebnis in einer neuen Excel-Datei oder
CSV
82         # result_df.to_excel('Aggregierte_Fallzahlen.xlsx', index=False)
83         # result_df.to_csv('Aggregierte_Fallzahlen.csv', index=False)

```

Listing 25: Skript Testfall 2 Prompt 2 Ausführung 4

```

1 import pandas as pd
2
3 def process_fallzahlen_excel(file_path):
4     """
5     Liest die Excel-Datei 'Fallzahlen.xlsx', verarbeitet die Daten aus allen
Sheets,
6     entfernt bestimmte LOR-Schlüssel, fasst die Daten zusammen, sortiert sie
nach
7     'Straftaten_insgesamt' und gibt den finalen DataFrame zurück.
8
9     :param file_path: Pfad zur Excel-Datei
10    :return: Gefilterter und sortierter Pandas DataFrame
11    """
12    # Lese die Excel-Datei
13    try:
14        xls = pd.ExcelFile(file_path)
15    except FileNotFoundError:
16        print(f"Die Datei {file_path} wurde nicht gefunden.")
17        return None
18    except Exception as e:
19        print(f"Ein Fehler ist aufgetreten: {e}")
20        return None

```

```

21
22 # Liste zur Speicherung der einzelnen DataFrames
23 df_list = []
24
25 # Iteriere über alle Sheets
26 for sheet_name in xls.sheet_names:
27     try:
28         df = pd.read_excel(xls, sheet_name=sheet_name)
29
30         # Überprüfe, ob die notwendigen Spalten vorhanden sind
31         required_columns = ['LOR-Schlüssel', 'Bezirke', '
Straftaten_insgesamt']
32         if not all(col in df.columns for col in required_columns):
33             print(f"Ein oder mehrere erforderliche Spalten fehlen im
Sheet '{sheet_name}'.")
34             continue
35
36         # Entferne die Zeilen mit unerwünschten LOR-Schlüsseln
37         df_filtered = df[~df['LOR-Schlüssel'].isin([999900, 999999])]
38
39         # Wähle nur die benötigten Spalten
40         df_selected = df_filtered[required_columns]
41
42         # Füge eine neue Spalte für das Sheet hinzu (optional, falls benö
tigt)
43         df_selected['Sheet'] = sheet_name
44
45         # Füge den DataFrame der Liste hinzu
46         df_list.append(df_selected)
47
48     except Exception as e:
49         print(f"Fehler beim Verarbeiten des Sheets '{sheet_name}': {e}")
50         continue
51
52 if not df_list:
53     print("Keine Daten zum Verarbeiten gefunden.")
54     return None
55
56 # Füge alle DataFrames zusammen
57 combined_df = pd.concat(df_list, ignore_index=True)
58
59 # Gruppiere nach 'LOR-Schlüssel' und 'Bezirke' und summiere '
Straftaten_insgesamt'
60 grouped_df = combined_df.groupby(['LOR-Schlüssel', 'Bezirke'], as_index=
False)['Straftaten_insgesamt'].sum()
61
62 # Sortiere nach 'Straftaten_insgesamt' absteigend
63 sorted_df = grouped_df.sort_values(by='Straftaten_insgesamt', ascending=
False).reset_index(drop=True)
64
65 return sorted_df
66

```

```

67 if __name__ == "__main__":
68     # Pfad zur Excel-Datei
69     excel_file_path = 'Fallzahlen.xlsx'
70
71     # Verarbeite die Excel-Datei und erhalte den finalen DataFrame
72     final_df = process_fallzahlen_excel(excel_file_path)
73
74     if final_df is not None:
75         # Zeige die ersten paar Zeilen des finalen DataFrames an
76         print(final_df)
77
78         # Optional: Speichere den finalen DataFrame in einer neuen Excel-
oder CSV-Datei
79         # final_df.to_excel('Zusammengefasste_Fallzahlen.xlsx', index=False)
80         # final_df.to_csv('Zusammengefasste_Fallzahlen.csv', index=False)

```

Listing 26: Skript Testfall 2 Prompt 2 Ausführung 5

```

1 import pandas as pd
2
3 def main():
4     # 1. Excel-Datei einlesen und alle Sheets als DataFrames speichern
5     excel_datei = 'Fallzahlen.xlsx'
6     try:
7         # Liest alle Sheets und speichert sie in einem Dictionary
8         sheets_dict = pd.read_excel(excel_datei, sheet_name=None)
9     except FileNotFoundError:
10         print(f"Die Datei '{excel_datei}' wurde nicht gefunden.")
11         return
12     except Exception as e:
13         print(f"Fehler beim Lesen der Excel-Datei: {e}")
14         return
15
16     # 2. DataFrames der einzelnen Sheets zusammenführen
17     df_liste = []
18     for sheet_name, df in sheets_dict.items():
19         # Überprüfen, ob die erforderlichen Spalten existieren
20         if 'LOR-Schlüssel' not in df.columns or 'Bezirke' not in df.columns:
21             print(f"Sheet '{sheet_name}' enthält nicht die erforderlichen
Spalten.")
22             continue
23
24         # Setzen von 'LOR-Schlüssel' und 'Bezirke' als Index
25         df = df.set_index(['LOR-Schlüssel', 'Bezirke'])
26         df_liste.append(df)
27
28     if not df_liste:
29         print("Keine gültigen Sheets zum Verarbeiten gefunden.")
30         return
31
32     # Alle DataFrames zusammenfügen und numerische Spalten akkumulieren
33     zusammengefuegt_df = pd.concat(df_liste)
34     # Gruppieren nach den Index-Spalten und summieren der numerischen Werte

```

```

35     akkumuliert_df = zusammengefuegt_df.groupby(level=['LOR-Schlüssel', '
    Bezirke']).sum().reset_index()
36
37     # 3. DataFrame sortieren nach 'Straftaten_insgesamt', spezielle Zeilen am
    Ende platzieren
38     # Definieren der speziellen LOR-Schlüssel
39     spezielle_keys = [999900, 999999]
40
41     # Trennen der regulären und speziellen Zeilen
42     regulär_df = akkumuliert_df[~akkumuliert_df['LOR-Schlüssel'].isin(
    spezielle_keys)]
43     spezielle_df = akkumuliert_df[akkumuliert_df['LOR-Schlüssel'].isin(
    spezielle_keys)]
44
45     # Sortieren der regulären Zeilen nach 'Straftaten_insgesamt' absteigend
46     sortiert_regulär_df = regulär_df.sort_values(by='Straftaten_insgesamt',
    ascending=False)
47
48     # Kombinieren der sortierten regulären Zeilen mit den speziellen Zeilen
49     endgültiges_df = pd.concat([sortiert_regulär_df, spezielle_df],
    ignore_index=True)
50
51     # 4. Sortiertes DataFrame zurückgeben (hier als Ausgabe angezeigt)
52     print(endgültiges_df)
53
54 if __name__ == '__main__':
55     main()

```

Listing 27: Skript Testfall 2 Prompt 3 Ausführung 1

```

1 import pandas as pd
2
3 def process_fallzahlen_excel(file_path='Fallzahlen.xlsx'):
4     """
5     Liest die Excel-Datei ein, führt die Sheets zusammen, akkumuliert die
    Straftaten,
6     sortiert das DataFrame nach 'Straftaten_insgesamt' und platziert
    spezielle
7     LOR-Schlüssel am Ende.
8
9     Parameters:
10    - file_path: str, der Pfad zur Excel-Datei.
11
12    Returns:
13    - pd.DataFrame: Das sortierte DataFrame.
14    """
15    # Schritt 1: Excel-Datei einlesen und alle Sheets als DataFrames
    speichern
16    try:
17        sheets = pd.read_excel(file_path, sheet_name=None)
18    except FileNotFoundError:
19        print(f"Die Datei {file_path} wurde nicht gefunden.")
20        return None

```

```

21     except Exception as e:
22         print(f"Fehler beim Einlesen der Datei: {e}")
23         return None
24
25     # Schritt 2: DataFrames zusammenführen und Straftaten akkumulieren
26     # Dabei werden alle Sheets kompiliert und numerische Werte summiert
27     try:
28         # Alle DataFrames zu einem einzigen DataFrame zusammenführen
29         df_all = pd.concat(sheets.values(), ignore_index=True)
30
31         # Gruppieren nach 'LOR-Schlüssel' und 'Bezirke' und numerische
32         # Spalten summieren
33         group_columns = ['LOR-Schlüssel', 'Bezirke']
34         df_grouped = df_all.groupby(group_columns, as_index=False).sum()
35     except KeyError as e:
36         print(f"Erforderliche Spalte fehlt: {e}")
37         return None
38     except Exception as e:
39         print(f"Fehler beim Zusammenführen der DataFrames: {e}")
40         return None
41
42     # Schritt 3: Sortieren nach 'Straftaten_insgesamt', spezielle LOR-Schlüssel am Ende
43     try:
44         # Definieren der speziellen LOR-Schlüssel
45         special_keys = [999900, 999999]
46
47         # Separieren der speziellen und normalen Zeilen
48         df_special = df_grouped[df_grouped['LOR-Schlüssel'].isin(special_keys)]
49         df_normal = df_grouped[~df_grouped['LOR-Schlüssel'].isin(special_keys)]
50
51         # Sortieren der normalen Zeilen nach 'Straftaten_insgesamt'
52         # absteigend
53         df_normal_sorted = df_normal.sort_values(by='Straftaten_insgesamt',
54         ascending=False)
55
56         # Zusammenführen der sortierten normalen Zeilen mit den speziellen
57         # Zeilen
58         df_final = pd.concat([df_normal_sorted, df_special], ignore_index=True)
59     except KeyError:
60         print("Die Spalte 'Straftaten_insgesamt' ist im DataFrame nicht
61         vorhanden.")
62         return None
63     except Exception as e:
64         print(f"Fehler beim Sortieren des DataFrames: {e}")
65         return None
66
67     # Schritt 4: Rückgabe des sortierten DataFrames
68     return df_final

```

```

64
65 if __name__ == "__main__":
66     # Verarbeiten der Excel-Datei und Erhalten des sortierten DataFrames
67     sorted_df = process_fallzahlen_excel('Fallzahlen.xlsx')
68
69     if sorted_df is not None:
70         # Anzeige der ersten paar Zeilen des sortierten DataFrames
71         print(sorted_df.head())
72
73         # Optional: Speichern des sortierten DataFrames in einer neuen Excel-
74         # Datei
75         # sorted_df.to_excel('Fallzahlen_sortiert.xlsx', index=False)

```

Listing 28: Skript Testfall 2 Prompt 3 Ausführung 2

```

1 import pandas as pd
2
3 def main():
4     # 1. Excel-Datei einlesen und alle Sheets als DataFrames speichern
5     excel_datei = 'Fallzahlen.xlsx'
6     sheets_dict = pd.read_excel(excel_datei, sheet_name=None)
7
8     # 2. DataFrames der einzelnen Sheets zusammenführen (joinen)
9     # Alle DataFrames vertikal concatenieren
10    concatenated_df = pd.concat(sheets_dict.values(), ignore_index=True)
11
12    # Gruppieren nach 'LOR-Schlüssel' und 'Bezirke' und die numerischen
13    # Spalten akkumulieren
14    gruppiert_df = concatenated_df.groupby(['LOR-Schlüssel', 'Bezirke'],
15    as_index=False).sum()
16
17    # 3. DataFrame sortieren nach 'Straftaten_insgesamt', dabei LOR-Schlüssel
18    # 999900 und 999999 am Ende lassen
19    # Definieren der auszuschließenden LOR-Schlüssel
20    ausschluss_keys = [999900, 999999]
21
22    # DataFrame ohne die auszuschließenden LOR-Schlüssel
23    zu_sortieren_df = gruppiert_df[~gruppiert_df['LOR-Schlüssel'].isin(
24    ausschluss_keys)]
25
26    # DataFrame mit den auszuschließenden LOR-Schlüsseln
27    ausgeschlossen_df = gruppiert_df[gruppiert_df['LOR-Schlüssel'].isin(
28    ausschluss_keys)]
29
30    # Sortieren nach 'Straftaten_insgesamt' in absteigender Reihenfolge
31    sortiert_df = zu_sortieren_df.sort_values(by='Straftaten_insgesamt',
32    ascending=False)
33
34    # Ausschließen Zeilen am Ende des sortierten DataFrames anhängen
35    final_df = pd.concat([sortiert_df, ausgeschlossen_df], ignore_index=True)
36
37    # 4. Sortiertes DataFrame zurückgeben (hier wird es angezeigt)
38    print(final_df)

```

```

33
34 if __name__ == '__main__':
35     main()

```

Listing 29: Skript Testfall 2 Prompt 3 Ausführung 3

```

1 import pandas as pd
2
3 def process_fallzahlen(excel_file):
4     # 1. Die Excel-Datei einlesen und die Sheets als DataFrames speichern.
5     # Alle Sheets werden in einem Dictionary gespeichert, wobei der Sheet-
6     # Name der Schlüssel ist.
7     xls = pd.ExcelFile(excel_file)
8     sheets_dict = {}
9     for sheet_name in xls.sheet_names:
10         df = pd.read_excel(xls, sheet_name=sheet_name)
11         sheets_dict[sheet_name] = df
12
13     # 2. Die DataFrames der einzelnen Sheets zusammen joinen,
14     # sodass pro Zeile der akkumulierte Wert der einzelnen Straftaten steht.
15     # Annahme: Alle DataFrames haben die selben 'LOR-Schlüssel' und 'Bezirke'
16     # Wir setzen 'LOR-Schlüssel' und 'Bezirke' als Index und summieren die
17     # numerischen Spalten.
18
19     # Liste der DataFrames
20     df_list = []
21     for df in sheets_dict.values():
22         # Setze 'LOR-Schlüssel' und 'Bezirke' als Index
23         df = df.set_index(['LOR-Schlüssel', 'Bezirke'])
24         df_list.append(df)
25
26     # Addiere alle DataFrames
27     # Dabei werden numerische Spalten summiert, nicht-numerische ignoriert
28     joined_df = pd.concat(df_list).groupby(['LOR-Schlüssel', 'Bezirke']).sum()
29     joined_df = joined_df.reset_index()
30
31     # 3. Das neue gejointe DataFrame nach der Spalte "Straftaten_insgesamt"
32     # sortieren.
33     # Zeilen mit den LOR-Schlüsseln 999900 und 999999 werden nicht in die
34     # Sortierung einbezogen
35     # und am Ende des DataFrames angehängt.
36
37     # Trenne die speziellen LOR-Schlüssel
38     special_keys = [999900, 999999]
39     normal_df = joined_df[~joined_df['LOR-Schlüssel'].isin(special_keys)]
40     special_df = joined_df[joined_df['LOR-Schlüssel'].isin(special_keys)]
41
42     # Sortiere das normale DataFrame nach 'Straftaten_insgesamt' absteigend
43     sorted_normal_df = normal_df.sort_values(by='Straftaten_insgesamt',
44                                               ascending=False)
45
46     # Füge die speziellen Zeilen am Ende hinzu

```



```

41     final_df = pd.concat([sorted_normal_df, special_df], ignore_index=True)
42
43     # 4. Das sortierte Pandas DataFrame zurückgeben.
44     return final_df
45
46 if __name__ == "__main__":
47     # Pfad zur Excel-Datei
48     excel_file = 'Fallzahlen.xlsx'
49
50     # Verarbeite die Excel-Datei
51     sorted_df = process_fallzahlen(excel_file)
52
53     # Optional: Zeige die ersten Zeilen des sortierten DataFrames an
54     print(sorted_df.head())
55
56     # Optional: Speichere das sortierte DataFrame in eine neue Excel-Datei
57     sorted_df.to_excel('Fallzahlen_sortiert.xlsx', index=False)

```

Listing 30: Skript Testfall 2 Prompt 3 Ausführung 4

```

1  import pandas as pd
2
3  def process_fallzahlen_excel(file_path='Fallzahlen.xlsx'):
4      """
5      Liest die Excel-Datei ein, verbindet die Sheets, sortiert das Ergebnis
6      nach
7      'Straftaten_insgesamt' und platziert die nicht zuordenbaren Einträge am
8      Ende.
9
10     Parameters:
11     - file_path: Pfad zur Excel-Datei (Standard: 'Fallzahlen.xlsx')
12
13     Returns:
14     - final_df: Sortiertes Pandas DataFrame
15     """
16     # Schritt 1: Alle Sheets der Excel-Datei einlesen
17     try:
18         excel_sheets = pd.read_excel(file_path, sheet_name=None)
19     except FileNotFoundError:
20         print(f"Datei '{file_path}' wurde nicht gefunden.")
21         return
22     except Exception as e:
23         print(f"Fehler beim Einlesen der Datei: {e}")
24         return
25
26     # Liste zur Speicherung der DataFrames
27     df_list = []
28
29     for sheet_name, df in excel_sheets.items():
30         # Sicherstellen, dass 'LOR-Schlüssel' und 'Bezirke' als Schlüssel
31         # vorhanden sind
32         if 'LOR-Schlüssel' not in df.columns or 'Bezirke' not in df.columns:
33             print(f"Sheet '{sheet_name}' enthält nicht die notwendigen

```

```

        Spalten.")
31         continue
32         df_list.append(df)
33
34     if not df_list:
35         print("Keine gültigen Sheets gefunden.")
36         return
37
38     # Schritt 2: DataFrames zusammenführen (joinen) auf 'LOR-Schlüssel' und '
    Bezirke'
39     merged_df = df_list[0]
40     for df in df_list[1:]:
41         merged_df = pd.merge(merged_df, df, on=['LOR-Schlüssel', 'Bezirke'],
    how='outer', suffixes=('', '_dup'))
42
43         # Entfernen von doppelten Spalten, falls vorhanden
44         dup_columns = [col for col in merged_df.columns if col.endswith('_dup
    ')]
45         if dup_columns:
46             merged_df.drop(columns=dup_columns, inplace=True)
47
48     # Schritt 3: Sortieren nach 'Straftaten_insgesamt', wobei 999900 und
    999999 am Ende stehen
49     # Zuerst sicherstellen, dass 'Straftaten_insgesamt' numerisch ist
50     merged_df['Straftaten_insgesamt'] = pd.to_numeric(merged_df['
    Straftaten_insgesamt'], errors='coerce')
51
52     # Trennen der DataFrames
53     districts_df = merged_df[~merged_df['LOR-Schlüssel'].isin([999900,
    999999])]
54     non_districts_df = merged_df[merged_df['LOR-Schlüssel'].isin([999900,
    999999])]
55
56     # Sortieren der Bezirke
57     sorted_districts_df = districts_df.sort_values(by='Straftaten_insgesamt',
    ascending=False)
58
59     # Kombinieren der sortierten Bezirke mit den nicht zuordenbaren Einträgen
60     final_df = pd.concat([sorted_districts_df, non_districts_df],
    ignore_index=True)
61
62     return final_df
63
64 if __name__ == "__main__":
65     # Pfad zur Excel-Datei
66     excel_file = 'Fallzahlen.xlsx'
67
68     # Verarbeitung der Excel-Datei
69     sorted_df = process_fallzahlen_excel(excel_file)
70
71     if sorted_df is not None:
72         # Ausgabe des sortierten DataFrames

```

```

73     print(sorted_df)
74
75     # Optional: Speichern des Ergebnisses in einer neuen Excel-Datei
76     # sorted_df.to_excel('Sorted_Fallzahlen.xlsx', index=False)

```

Listing 31: Skript Testfall 2 Prompt 3 Ausführung 5

### 8.2.3 Testfall 3

```

1  import pandas as pd
2
3  def berechne_strafttaten_anteil(excel_datei, sheet_name, ausgabe_datei):
4      # Excel-Datei einlesen
5      df = pd.read_excel(excel_datei, sheet_name=sheet_name)
6
7      # Anzeigen der Daten zur Überprüfung (optional)
8      # print(df.head())
9
10     # Identifizieren der Gesamtzeile für Berlin
11     # Annahme: Die Gesamtzeile hat 'Bezirke' als "Berlin (PKS gesamt)"
12     gesamt_row = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
13
14     if gesamt_row.empty:
15         # Falls keine explizite Gesamtzeile vorhanden ist, summiere alle
16         # relevanten Bezirke
17         # Annahme: Die Gesamtzeile hat einen LOR-Schlüssel von 999999
18         gesamt_row = df[df['LOR-Schlüssel'] == 999999]
19
20     if gesamt_row.empty:
21         raise ValueError("Gesamtdaten für Berlin nicht gefunden.")
22
23     # Gesamtzahl der Straftaten in Berlin
24     gesamt_strafttaten = gesamt_row['Straftaten_insgesamt'].values[0]
25
26     # Filtere nur die Bezirke, die zu Berlin gehören und nicht die
27     # Gesamtzeile oder nicht zuzuordnend
28     # Annahme: 'Stadtgebiet Berlin, nicht zuzuordnen' und 'Berlin (PKS gesamt)'
29     # sollen ausgeschlossen werden
30     relevante_bezirke = df[~df['Bezirke'].isin(['Berlin (PKS gesamt)', '
31     Stadtgebiet Berlin, nicht zuzuordnen'])]
32
33     # Berechne den prozentualen Anteil
34     relevante_bezirke = relevante_bezirke.copy() # Um eine Kopie zu
35     erstellen
36     relevante_bezirke['Prozentualer Anteil (%)'] = (relevante_bezirke['
37     Straftaten_insgesamt'] / gesamt_strafttaten) * 100
38
39     # Optional: Runden auf zwei Dezimalstellen
40     relevante_bezirke['Prozentualer Anteil (%)'] = relevante_bezirke['
41     Prozentualer Anteil (%)'].round(2)
42
43     # Sortiere die Bezirke nach dem prozentualen Anteil absteigend

```

```

37     relevante_bezirke = relevante_bezirke.sort_values(by='Prozentualer Anteil
    (%)', ascending=False)
38
39     # Auswahl der relevanten Spalten für die Ausgabe
40     ausgabe_df = relevante_bezirke[['Bezirke', 'Straftaten_insgesamt', '
    Prozentualer Anteil (%)']]
41
42     # Ausgabe in eine neue Excel-Datei
43     ausgabe_df.to_excel(ausgabe_datei, index=False)
44
45     print(f"Die prozentualen Anteile wurden erfolgreich berechnet und in '{
    ausgabe_datei}' gespeichert.")
46
47 if __name__ == "__main__":
48     # Pfad zur Eingabe-Excel-Datei
49     input_excel = 'Fallzahlen.xlsx'
50
51     # Name des Sheets
52     sheet = 'Fallzahlen_2023'
53
54     # Name der Ausgabedatei
55     output_excel = 'Straftaten_Anteil_Bezirke.xlsx'
56
57     # Funktion aufrufen
58     berechne_straftaten_anteil(input_excel, sheet, output_excel)

```

Listing 32: Skript Testfall 3 Prompt 1 Ausführung 1

```

1 import pandas as pd
2
3 # Pfad zur Excel-Datei
4 excel_file = 'Fallzahlen.xlsx'
5
6 # Name des Sheets
7 sheet_name = 'Fallzahlen_2023'
8
9 # Lesen des Excel-Sheets
10 df = pd.read_excel(excel_file, sheet_name=sheet_name)
11
12 # Annahmen, dass die Gesamtzahl in der Zeile mit 'Berlin (PKS gesamt)' steht
13 # Suchen der Gesamtzeile
14 total_row = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
15
16 if total_row.empty:
17     raise ValueError("Die Gesamtzeile 'Berlin (PKS gesamt)' wurde nicht
    gefunden.")
18
19 # Extrahieren der Gesamtzahl der Straftaten
20 total_crimes = total_row['Straftaten_insgesamt'].values[0]
21
22 # Filtern der Bezirke (ohne Gesamtzeile)
23 districts_df = df[df['Bezirke'] != 'Berlin (PKS gesamt)'].copy()
24

```

```

25 # Berechnen des prozentualen Anteils
26 districts_df['Prozentualer_Anteil (%)'] = (districts_df['Straftaten_insgesamt
    ']/ total_crimes) * 100
27
28 # Optional: Runden auf zwei Dezimalstellen
29 districts_df['Prozentualer_Anteil (%)'] = districts_df['Prozentualer_Anteil
    (%)'].round(2)
30
31 # Anzeigen der Ergebnisse
32 print(districts_df[['Bezirke', 'Straftaten_insgesamt', 'Prozentualer_Anteil
    (%)']])
33
34 # Optional: Speichern der Ergebnisse in eine neue Excel-Datei
35 output_file = 'Straftaten_Prozentual_Anteil.xlsx'
36 districts_df.to_excel(output_file, sheet_name='Prozentuale Anteile', index=
    False)
37 print(f"\nDie Ergebnisse wurden in '{output_file}' gespeichert.")

```

Listing 33: Skript Testfall 3 Prompt 1 Ausführung 2

```

1 import pandas as pd
2
3 # Pfad zur Excel-Datei
4 excel_datei = 'Fallzahlen.xlsx'
5
6 # Name des Sheets
7 sheet_name = 'Fallzahlen_2023'
8
9 # Lesen der Excel-Datei
10 try:
11     df = pd.read_excel(excel_datei, sheet_name=sheet_name)
12 except FileNotFoundError:
13     print(f"Die Datei '{excel_datei}' wurde nicht gefunden.")
14     exit(1)
15 except Exception as e:
16     print(f"Fehler beim Lesen der Excel-Datei: {e}")
17     exit(1)
18
19 # Anzeigen der geladenen Daten (optional)
20 #print(df.head())
21
22 # Herausfinden der Gesamtstraftaten in Berlin
23 # Annahme: Die Gesamtstraftaten stehen in der Zeile mit 'Berlin (PKS gesamt)'
24 gesamt_row = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
25
26 if gesamt_row.empty:
27     print("Die Zeile 'Berlin (PKS gesamt)' wurde nicht gefunden.")
28     exit(1)
29
30 gesamt_straftaten = gesamt_row['Straftaten_insgesamt'].values[0]
31
32 # Filtern der Bezirke (Ausschluss von Gesamtzeilen)
33 bezirke_df = df[~df['Bezirke'].isin(['Berlin (PKS gesamt)', 'Stadtgebiet

```

```

        Berlin, nicht zuzuordnen']]])
34
35 # Berechnung des prozentualen Anteils
36 bezirke_df = bezirke_df.copy() # Vermeidung von SettingWithCopyWarning
37 bezirke_df['Prozentualer_Anteil (%)'] = (bezirke_df['Straftaten_insgesamt'] /
    gesamt_straftaten) * 100
38
39 # Optional: Runden auf zwei Dezimalstellen
40 bezirke_df['Prozentualer_Anteil (%)'] = bezirke_df['Prozentualer_Anteil (%)'
    ].round(2)
41
42 # Sortieren nach Prozentualem Anteil absteigend (optional)
43 bezirke_df = bezirke_df.sort_values(by='Prozentualer_Anteil (%)', ascending=
    False)
44
45 # Ergebnis anzeigen
46 ergebnis = bezirke_df[['Bezirke', 'Straftaten_insgesamt', '
    Prozentualer_Anteil (%)']]
47 print(ergebnis)
48
49 # Optional: Ergebnis in eine neue Excel-Datei speichern
50 # ergebnis.to_excel('Prozentuale_Anteile_Straftaten.xlsx', index=False)

```

Listing 34: Skript Testfall 3 Prompt 1 Ausführung 3

```

1 import pandas as pd
2
3 def berechne_prozentuale_anteile(dateipfad: str, sheet_name: str = '
    Fallzahlen_2023'):
4     """
5     Berechnet den prozentualen Anteil der Straftaten jedes Bezirks an den
    gesamten Straftaten in Berlin.
6
7     :param dateipfad: Pfad zur Excel-Datei 'Fallzahlen.xlsx'.
8     :param sheet_name: Name des Sheets in der Excel-Datei. Standard ist '
    Fallzahlen_2023'.
9     :return: DataFrame mit Bezirken und deren prozentualen Anteilen.
10    """
11    try:
12        # Excel-Datei lesen
13        df = pd.read_excel(dateipfad, sheet_name=sheet_name)
14    except FileNotFoundError:
15        print(f"Die Datei {dateipfad} wurde nicht gefunden.")
16        return
17    except Exception as e:
18        print(f"Ein Fehler ist beim Lesen der Excel-Datei aufgetreten: {e}")
19        return
20
21    # Gesamtstraftaten für Berlin abrufen
22    total_crimes_row = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
23    if not total_crimes_row.empty:
24        total_crimes = total_crimes_row['Straftaten_insgesamt'].values[0]
25    else:

```

```

26     # Falls 'Berlin (PKS gesamt)' nicht vorhanden ist, summiere die
    Bezirke
27     exclude_bezirke = ['Berlin (PKS gesamt)', 'Stadtgebiet Berlin, nicht
    zuzuordnen']
28     total_crimes = df.loc[~df['Bezirke'].isin(exclude_bezirke), '
    Straftaten_insgesamt'].sum()
29
30     # Bezirke filtern (ausschließen: Gesamtdaten und nicht zuordenbare
    Gebiete)
31     exclude_bezirke = ['Berlin (PKS gesamt)', 'Stadtgebiet Berlin, nicht
    zuzuordnen']
32     bezirke_df = df.loc[~df['Bezirke'].isin(exclude_bezirke)].copy()
33
34     # Prozentualen Anteil berechnen
35     bezirke_df['Prozentualer_Anteil'] = (bezirke_df['Straftaten_insgesamt'] /
    total_crimes) * 100
36
37     # Optional: Auf zwei Dezimalstellen runden
38     bezirke_df['Prozentualer_Anteil'] = bezirke_df['Prozentualer_Anteil'].
    round(2)
39
40     # Ergebnis anzeigen
41     print("Prozentualer Anteil der Straftaten nach Bezirk:")
42     print(bezirke_df[['Bezirke', 'Straftaten_insgesamt', 'Prozentualer_Anteil
    ']])
43
44     # Optional: Ergebnis in eine neue Excel-Datei speichern
45     output_datei = 'Prozentuale_Anteile_Straftaten.xlsx'
46     try:
47         bezirke_df[['Bezirke', 'Straftaten_insgesamt', 'Prozentualer_Anteil'
    ]].to_excel(output_datei, index=False)
48         print(f"\nDie Ergebnisse wurden in '{output_datei}' gespeichert.")
49     except Exception as e:
50         print(f"Ein Fehler ist beim Speichern der Ergebnisdatei aufgetreten:
    {e}")
51
52 if __name__ == "__main__":
53     # Pfad zur Excel-Datei
54     dateipfad = 'Fallzahlen.xlsx'
55
56     # Prozentuale Anteile berechnen
57     berechne_prozentuale_anteile(dateipfad)

```

Listing 35: Skript Testfall 3 Prompt 1 Ausführung 4

```

1 import pandas as pd
2
3 def berechne_prozentualer_anteil_straftaten():
4     try:
5         # Excel-Datei laden
6         dateipfad = 'Fallzahlen.xlsx' # Pfad zur Excel-Datei
7         sheet_name = 'Fallzahlen_2023' # Name des Sheets
8

```

```

9         df = pd.read_excel(dateipfad, sheet_name=sheet_name)
10
11         # Überprüfen, ob die benötigten Spalten vorhanden sind
12         erforderliche_spalten = ['Bezirke', 'Straftaten_insgesamt']
13         fehlende_spalten = [spalte for spalte in erforderliche_spalten if
14                             spalte not in df.columns]
15         if fehlende_spalten:
16             raise ValueError(f"Die folgenden benötigten Spalten fehlen im
17                               Excel-Sheet: {fehlende_spalten}")
18
19         # Gesamte Straftaten aus der Gesamtzeile holen
20         gesamt_row = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
21         if not gesamt_row.empty:
22             gesamt_straftaten = gesamt_row['Straftaten_insgesamt'].iloc[0]
23             print(f"Gesamte Straftaten in Berlin: {gesamt_straftaten}")
24         else:
25             # Falls die Gesamtzeile nicht vorhanden ist, summiere alle
26             # Bezirkswerte
27             gesamt_straftaten = df[~df['Bezirke'].isin(['Stadtgebiet Berlin,
28                                                         nicht zuzuordnen'])]['Straftaten_insgesamt'].sum()
29             print(f"Gesamte Straftaten in Berlin (berechnet): {
30                   gesamt_straftaten}")
31
32         # Bezirke ausschließen, die nicht zugeordnet werden können oder die
33         # Gesamtzeile sind
34         bezirke_df = df[~df['Bezirke'].isin(['Berlin (PKS gesamt)', '
35                                               Stadtgebiet Berlin, nicht zuzuordnen'])]
36
37         # Prozentualen Anteil berechnen
38         bezirke_df = bezirke_df.copy() # Kopie erstellen, um
39         SettingWithCopyWarning zu vermeiden
40         bezirke_df['Prozentualer_Anteil (%)'] = (bezirke_df['
41                                                   Straftaten_insgesamt'] / gesamt_straftaten) * 100
42
43         # Prozentualen Anteil auf 2 Dezimalstellen runden
44         bezirke_df['Prozentualer_Anteil (%)'] = bezirke_df['
45                                                   Prozentualer_Anteil (%)'].round(2)
46
47         # Ergebnis anzeigen
48         print("\nProzentualer Anteil der Straftaten pro Bezirk:")
49         print(bezirke_df[['Bezirke', 'Straftaten_insgesamt', '
50                           Prozentualer_Anteil (%)']])
51
52         # Optional: Ergebnis in eine neue Excel-Datei speichern
53         ausgabe_datei = 'Straftaten_Prozentual.xlsx'
54         bezirke_df[['Bezirke', 'Straftaten_insgesamt', 'Prozentualer_Anteil
55                     (%)']].to_excel(ausgabe_datei, index=False)
56         print(f"\nDas Ergebnis wurde in '{ausgabe_datei}' gespeichert.")
57
58     except FileNotFoundError:
59         print("Fehler: Die Datei 'Fallzahlen.xlsx' wurde nicht gefunden.
60               Bitte überprüfe den Dateipfad.")

```



```

48     except ValueError as ve:
49         print(f"Fehler: {ve}")
50     except Exception as e:
51         print(f"Ein unerwarteter Fehler ist aufgetreten: {e}")
52
53 if __name__ == "__main__":
54     berechne_prozentualer_anteil_straftaten()

```

Listing 36: Skript Testfall 3 Prompt 1 Ausführung 5

```

1  import pandas as pd
2
3  # Pfad zur Excel-Datei
4  excel_datei = 'Fallzahlen.xlsx'
5
6  # Name des Sheets
7  sheet_name = 'Fallzahlen_2023'
8
9  # Lesen der Excel-Datei
10 df = pd.read_excel(excel_datei, sheet_name=sheet_name)
11
12 # Überprüfen, ob die notwendigen Spalten vorhanden sind
13 notwendige_spalten = ['Bezirke', 'Straftaten_insgesamt']
14 for spalte in notwendige_spalten:
15     if spalte not in df.columns:
16         raise ValueError(f"Die Spalte '{spalte}' fehlt in der Excel-Datei.")
17
18 # Extrahieren der Gesamtanzahl von ganz Berlin
19 gesamt_row = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
20
21 if gesamt_row.empty:
22     raise ValueError("Die Gesamtzeile 'Berlin (PKS gesamt)' wurde nicht gefunden.")
23
24 gesamt_straftaten = gesamt_row['Straftaten_insgesamt'].values[0]
25
26 # Entfernen der Gesamtzeile aus dem DataFrame
27 bezirk_df = df[df['Bezirke'] != 'Berlin (PKS gesamt)'].copy()
28
29 # Berechnung des prozentualen Anteils
30 bezirk_df['Prozent_Straftaten_insgesamt'] = (bezirk_df['Straftaten_insgesamt']
31     ] / gesamt_straftaten) * 100
32
33 # Optional: Runde die Prozentwerte auf zwei Dezimalstellen
34 bezirk_df['Prozent_Straftaten_insgesamt'] = bezirk_df['
35     Prozent_Straftaten_insgesamt'].round(2)
36
37 # Ausgabe des Ergebnisses
38 print(bezirk_df[['Bezirke', 'Straftaten_insgesamt', '
39     Prozent_Straftaten_insgesamt']])
40
41 # Optional: Speichern in einer neuen Excel-Datei
42 ausgabe_datei = 'Fallzahlen_mit_Prozentsatz.xlsx'

```

```

40 bezirk_df.to_excel(ausgabe_datei, index=False)
41 print(f"\nDie Ergebnisse wurden in '{ausgabe_datei}' gespeichert.")

```

Listing 37: Skript Testfall 3 Prompt 2 Ausführung 1

```

1  import pandas as pd
2
3  # Pfad zur Excel-Datei
4  excel_datei = 'Fallzahlen.xlsx'
5
6  # Name des Sheets
7  sheet_name = 'Fallzahlen_2023'
8
9  # Laden der Excel-Datei
10 try:
11     df = pd.read_excel(excel_datei, sheet_name=sheet_name)
12 except FileNotFoundError:
13     print(f"Die Datei '{excel_datei}' wurde nicht gefunden.")
14     exit(1)
15 except Exception as e:
16     print(f"Ein Fehler ist aufgetreten: {e}")
17     exit(1)
18
19 # Sicherstellen, dass die notwendigen Spalten vorhanden sind
20 erforderliche_spalten = ['Bezirke', 'Straftaten_insgesamt']
21 for spalte in erforderliche_spalten:
22     if spalte not in df.columns:
23         print(f"Die Spalte '{spalte}' fehlt in der Excel-Datei.")
24         exit(1)
25
26 # Extrahieren der Gesamtanzahl von ganz Berlin
27 gesamt_zeile = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
28
29 if gesamt_zeile.empty:
30     print("Die Zeile 'Berlin (PKS gesamt)' wurde nicht gefunden.")
31     exit(1)
32
33 # Annahme: Es gibt genau eine Zeile mit 'Berlin (PKS gesamt)'
34 gesamt_straftaten = gesamt_zeile['Straftaten_insgesamt'].values[0]
35
36 if gesamt_straftaten == 0:
37     print("Die Gesamtanzahl der Straftaten für Berlin ist 0. Division durch Null ist nicht möglich.")
38     exit(1)
39
40 # Filtern der Bezirke (ohne die Gesamtzeile)
41 bezirke_df = df[df['Bezirke'] != 'Berlin (PKS gesamt)'].copy()
42
43 # Berechnung des prozentualen Anteils
44 bezirke_df['Prozentualer_Anteil (%)'] = (bezirke_df['Straftaten_insgesamt'] /
45     gesamt_straftaten) * 100
46
47 # Optional: Runden auf zwei Dezimalstellen

```

```

47 bezirke_df['Prozentualer_Anteil (%)'] = bezirke_df['Prozentualer_Anteil (%)'
    ].round(2)
48
49 # Sortieren nach dem prozentualen Anteil absteigend
50 bezirke_df.sort_values(by='Prozentualer_Anteil (%)', ascending=False, inplace
    =True)
51
52 # Ergebnis anzeigen
53 print(bezirke_df[['Bezirke', 'Straftaten_insgesamt', 'Prozentualer_Anteil (%)
    ']])
54
55 # Optional: Speichern in eine neue Excel-Datei
56 output_datei = 'Fallzahlen_prozentual.xlsx'
57 try:
58     bezirke_df.to_excel(output_datei, sheet_name='Prozentualer_Anteil', index
    =False)
59     print(f"\nDie Ergebnisse wurden in '{output_datei}' gespeichert.")
60 except Exception as e:
61     print(f"Fehler beim Speichern der Datei: {e}")

```

Listing 38: Skript Testfall 3 Prompt 2 Ausführung 2

```

1 import pandas as pd
2
3 # Pfad zur Excel-Datei
4 excel_datei = 'Fallzahlen.xlsx'
5
6 # Name des Sheets
7 sheet_name = 'Fallzahlen_2023'
8
9 # Lesen des Excel-Sheets
10 try:
11     df = pd.read_excel(excel_datei, sheet_name=sheet_name)
12 except FileNotFoundError:
13     print(f"Die Datei '{excel_datei}' wurde nicht gefunden.")
14     exit(1)
15 except Exception as e:
16     print(f"Ein Fehler ist aufgetreten beim Lesen der Excel-Datei: {e}")
17     exit(1)
18
19 # Überprüfen, ob die notwendigen Spalten vorhanden sind
20 if 'Bezirke' not in df.columns or 'Straftaten_insgesamt' not in df.columns:
21     print("Die erforderlichen Spalten 'Bezirke' oder 'Straftaten_insgesamt'
    fehlen in der Tabelle.")
22     exit(1)
23
24 # Finden des Gesamtwerts für Berlin
25 gesamt_zeile = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
26
27 if gesamt_zeile.empty:
28     print("Die Gesamtzeile 'Berlin (PKS gesamt)' wurde nicht gefunden.")
29     exit(1)
30

```

```

31 gesamt_straftaten = gesamt_zeile['Straftaten_insgesamt'].values[0]
32
33 # Entfernen der Gesamtzeile aus den Bezirken
34 bezirke_df = df[df['Bezirke'] != 'Berlin (PKS gesamt)'].copy()
35
36 # Berechnen des prozentualen Anteils
37 bezirke_df['Prozentualer_Anteil'] = (bezirke_df['Straftaten_insgesamt'] /
    gesamt_straftaten) * 100
38
39 # Optional: Runden auf zwei Dezimalstellen
40 bezirke_df['Prozentualer_Anteil'] = bezirke_df['Prozentualer_Anteil'].round
    (2)
41
42 # Anzeigen der Ergebnisse
43 print(bezirke_df[['Bezirke', 'Straftaten_insgesamt', 'Prozentualer_Anteil']])
44
45 # Speichern der Ergebnisse in einer neuen Excel-Datei
46 ausgabe_datei = 'Fallzahlen_Prozente.xlsx'
47 try:
48     bezirke_df.to_excel(ausgabe_datei, sheet_name='Prozentuale_Fallzahlen',
    index=False)
49     print(f"\nDie prozentualen Anteile wurden in '{ausgabe_datei}'
    gespeichert.")
50 except Exception as e:
51     print(f"Ein Fehler ist aufgetreten beim Speichern der Excel-Datei: {e}")

```

Listing 39: Skript Testfall 3 Prompt 2 Ausführung 3

```

1 import pandas as pd
2
3 # Pfad zur Excel-Datei und Name des Sheets
4 excel_datei = 'Fallzahlen.xlsx'
5 blatt_name = 'Fallzahlen_2023'
6
7 # Lade die Daten aus der Excel-Datei
8 df = pd.read_excel(excel_datei, sheet_name=blatt_name)
9
10 # Finde die Gesamtzahl der Straftaten für ganz Berlin
11 berlin_gesamtreihe = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
12 if berlin_gesamtreihe.empty:
13     raise ValueError("Die Zeile 'Berlin (PKS gesamt)' wurde in den Daten
    nicht gefunden.")
14
15 berlin_gesamt = berlin_gesamtreihe['Straftaten_insgesamt'].iloc[0]
16
17 # Entferne die Gesamtzeile aus dem DataFrame
18 bezirke_df = df[df['Bezirke'] != 'Berlin (PKS gesamt)'].copy()
19
20 # Berechne den prozentualen Anteil der Straftaten für jeden Bezirk
21 bezirke_df['Prozentualer_Anteil'] = ((bezirke_df['Straftaten_insgesamt'] /
    berlin_gesamt) * 100).round(2)
22
23 # Ausgabe der Ergebnisse

```

```

24 print(bezirke_df[['Bezirke', 'Straftaten_insgesamt', 'Prozentualer_Anteil']])
25
26 # Speichere das Ergebnis in eine neue Excel-Datei
27 bezirke_df.to_excel('Fallzahlen_mit_Prozentanteilen.xlsx', index=False)

```

Listing 40: Skript Testfall 3 Prompt 2 Ausführung 4

```

1 import pandas as pd
2
3 # Pfad zur Excel-Datei und Name des Sheets
4 excel_datei = 'Fallzahlen.xlsx'
5 sheet_name = 'Fallzahlen_2023'
6
7 # Excel-Datei einlesen
8 df = pd.read_excel(excel_datei, sheet_name=sheet_name)
9
10 # Gesamtanzahl der Straftaten in Berlin extrahieren
11 # Annahme: Der Eintrag für "Berlin (PKS gesamt)" ist eindeutig
12 gesamt_strafaten = df.loc[df['Bezirke'] == 'Berlin (PKS gesamt)', '
    Straftaten_insgesamt'].values[0]
13
14 # Entfernen der Gesamtzeile, um nur die Bezirke zu betrachten
15 df_bezirke = df[df['Bezirke'] != 'Berlin (PKS gesamt)'].copy()
16
17 # Berechnung des prozentualen Anteils der Straftaten für jeden Bezirk
18 df_bezirke['Prozent_Straftaten'] = (df_bezirke['Straftaten_insgesamt'] /
    gesamt_strafaten) * 100
19
20 # Optional: Sortieren nach dem prozentualen Anteil
21 df_bezirke.sort_values(by='Prozent_Straftaten', ascending=False, inplace=True)
22
23 # Ergebnis anzeigen
24 print(df_bezirke[['Bezirke', 'Straftaten_insgesamt', 'Prozent_Straftaten']])
25
26 # Optional: Ergebnisse in eine neue Excel-Datei speichern
27 ausgabe_datei = 'Fallzahlen_mit_Prozent.xlsx'
28 df_bezirke.to_excel(ausgabe_datei, index=False)
29 print(f"\nDie Ergebnisse wurden in '{ausgabe_datei}' gespeichert.")

```

Listing 41: Skript Testfall 3 Prompt 2 Ausführung 5

```

1 import pandas as pd
2
3 def main():
4     # 1. Excel-Datei einlesen
5     excel_datei = 'Fallzahlen.xlsx'
6     sheet_name = 'Fallzahlen_2023'
7
8     try:
9         df = pd.read_excel(excel_datei, sheet_name=sheet_name)
10        print(f"Excel-Datei '{excel_datei}' erfolgreich eingelesen.")
11    except FileNotFoundError:

```

```

12     print(f"Fehler: Die Datei '{excel_datei}' wurde nicht gefunden.")
13     return
14 except Exception as e:
15     print(f"Ein Fehler ist beim Einlesen der Excel-Datei aufgetreten: {e}")
16     return
17
18 # 2. Tabelle als Pandas DataFrame speichern
19 # (bereits durch das Einlesen mit pandas erledigt)
20
21 # 3. Überprüfen, ob die notwendigen Spalten vorhanden sind
22 erforderliche_spalten = ['Bezirke', 'Straftaten_insgesamt']
23 fehlende_spalten = [spalte for spalte in erforderliche_spalten if spalte
24 not in df.columns]
25
26 if fehlende_spalten:
27     print(f"Fehler: Fehlende Spalten in der Excel-Datei: {
28 fehlende_spalten}")
29     return
30 else:
31     print("Alle erforderlichen Spalten sind vorhanden.")
32
33 # 4. Gesamtzahl der Straftaten für ganz Berlin finden
34 gesamt_row = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
35
36 if gesamt_row.empty:
37     print("Fehler: Die Zeile mit 'Berlin (PKS gesamt)' wurde nicht
38 gefunden.")
39     return
40
41 gesamt_straftaten = gesamt_row['Straftaten_insgesamt'].values[0]
42 print(f"Gesamtzahl der Straftaten für Berlin: {gesamt_straftaten}")
43
44 # 5. Prozentualen Anteil der einzelnen Bezirke berechnen
45 # Ausschließen der Gesamtzeile für die Berechnung
46 df_bezirke = df[df['Bezirke'] != 'Berlin (PKS gesamt)'].copy()
47
48 if df_bezirke.empty:
49     print("Fehler: Es gibt keine Bezirke nach dem Ausschluss von 'Berlin
50 (PKS gesamt)'")
51     return
52
53 # Prozentualer Anteil berechnen
54 df_bezirke['Prozentualer_Anteil (%)'] = (df_bezirke['Straftaten_insgesamt']
55 / gesamt_straftaten) * 100
56
57 # Optional: Runden auf zwei Dezimalstellen
58 df_bezirke['Prozentualer_Anteil (%)'] = df_bezirke['Prozentualer_Anteil
59 (%)'].round(2)
60
61 print("Prozentualer Anteil der Straftaten pro Bezirk berechnet.")

```

```

57 # 6. Ergebnis als DataFrame zurückgeben
58 # Hier zeigen wir das Ergebnis an. Sie können es auch in eine Datei
    speichern.
59 ergebnis_df = df_bezirke[['Bezirke', 'Straftaten_insgesamt', '
    Prozentualer_Anteil (%)']]
60 print("\nErgebnis DataFrame:")
61 print(ergebnis_df.to_string(index=False))
62
63 # Optional: Ergebnis in eine neue Excel-Datei speichern
64 ausgabe_datei = 'Fallzahlen_prozentual.xlsx'
65 try:
66     ergebnis_df.to_excel(ausgabe_datei, index=False)
67     print(f"\nErgebnis erfolgreich in '{ausgabe_datei}' gespeichert.")
68 except Exception as e:
69     print(f"Ein Fehler ist beim Speichern der Ergebnisdatei aufgetreten:
    {e}")
70
71 if __name__ == "__main__":
72     main()

```

Listing 42: Skript Testfall 3 Prompt 3 Ausführung 1

```

1 import pandas as pd
2
3 def verarbeite_fallzahlen(dateipfad: str, sheet_name: str = 'Fallzahlen_2023'
    ) -> pd.DataFrame:
4     """
5     Liest eine Excel-Datei ein, überprüft notwendige Spalten, berechnet
6     den prozentualen Anteil der Straftaten pro Bezirk und gibt das Ergebnis
        als DataFrame zurück.
7
8     :param dateipfad: Pfad zur Excel-Datei 'Fallzahlen.xlsx'.
9     :param sheet_name: Name des Sheets in der Excel-Datei. Standard ist '
        Fallzahlen_2023'.
10    :return: DataFrame mit Bezirken, Straftaten insgesamt und prozentualen
        Anteil.
11    """
12    try:
13        # 1. Excel-Datei einlesen
14        df = pd.read_excel(dateipfad, sheet_name=sheet_name)
15        print("Excel-Datei erfolgreich eingelesen.")
16    except FileNotFoundError:
17        raise FileNotFoundError(f"Die Datei '{dateipfad}' wurde nicht
        gefunden.")
18    except Exception as e:
19        raise Exception(f"Fehler beim Einlesen der Excel-Datei: {e}")
20
21    # 2. Die Tabelle als Pandas DataFrame gespeichert (bereits in 'df')
22
23    # 3. Überprüfen, ob die notwendigen Spalten vorhanden sind
24    erforderliche_spalten = ['Bezirke', 'Straftaten_insgesamt']
25    fehlende_spalten = [spalte for spalte in erforderliche_spalten if spalte
        not in df.columns]

```

```

26     if fehlende_spalten:
27         raise ValueError(f"Die folgenden erforderlichen Spalten fehlen in der
           Tabelle: {fehlende_spalten}")
28     else:
29         print("Alle erforderlichen Spalten sind vorhanden.")
30
31     # 4. Gesamtzahl der Straftaten für ganz Berlin finden
32     gesamt_row = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
33     if gesamt_row.empty:
34         raise ValueError("Die Zeile mit 'Berlin (PKS gesamt)' wurde nicht
           gefunden.")
35
36     gesamt_straftaten = gesamt_row['Straftaten_insgesamt'].iloc[0]
37     print(f"Gesamtzahl der Straftaten in Berlin: {gesamt_straftaten}")
38
39     if gesamt_straftaten == 0:
40         raise ValueError("Die Gesamtzahl der Straftaten ist 0, kann keine
           Prozente berechnen.")
41
42     # 5. Prozentualen Anteil der einzelnen Bezirke berechnen
43     # Ausschließen der Gesamtzeile von den einzelnen Bezirken
44     einzel_bezirke = df[df['Bezirke'] != 'Berlin (PKS gesamt)'].copy()
45
46     # Berechnung des prozentualen Anteils
47     einzel_bezirke['Prozentualer_Anteil'] = (einzel_bezirke['
           Straftaten_insgesamt'] / gesamt_straftaten) * 100
48
49     # Optional: Auf zwei Dezimalstellen runden
50     einzel_bezirke['Prozentualer_Anteil'] = einzel_bezirke['
           Prozentualer_Anteil'].round(2)
51
52     # 6. Ergebnis als DataFrame zurückgeben
53     ergebnis = einzel_bezirke[['Bezirke', 'Straftaten_insgesamt', '
           Prozentualer_Anteil']]
54
55     return ergebnis
56
57 if __name__ == "__main__":
58     dateipfad = 'Fallzahlen.xlsx' # Pfad zur Excel-Datei anpassen, falls
           erforderlich
59     try:
60         ergebnis_df = verarbeite_fallzahlen(dateipfad)
61         print("\nProzentualer Anteil der Straftaten pro Bezirk in Berlin:")
62         print(ergebnis_df.to_string(index=False))
63     except Exception as e:
64         print(f"Ein Fehler ist aufgetreten: {e}")

```

Listing 43: Skript Testfall 3 Prompt 3 Ausführung 2

```

1 import pandas as pd
2
3 def calculate_bezirk_straftaten_percentage(excel_file: str, sheet_name: str =
           'Fallzahlen_2023') -> pd.DataFrame:

```



```

4     """
5     Liest eine Excel-Datei ein und berechnet den prozentualen Anteil der
6     Straftaten_insgesamt für jeden Bezirk im Vergleich zu ganz Berlin.
7
8     Parameters:
9     - excel_file: Pfad zur Excel-Datei.
10    - sheet_name: Name des Sheets in der Excel-Datei (standardmäßig '
11    Fallzahlen_2023').
12
13    Returns:
14    - Ein Pandas DataFrame mit den Bezirken und ihrem prozentualen Anteil.
15    """
16
17    try:
18        # 1. Excel-Datei einlesen
19        df = pd.read_excel(excel_file, sheet_name=sheet_name)
20        print("Excel-Datei erfolgreich eingelesen.")
21    except FileNotFoundError:
22        raise FileNotFoundError(f"Die Datei '{excel_file}' wurde nicht
23        gefunden.")
24    except Exception as e:
25        raise Exception(f"Fehler beim Einlesen der Excel-Datei: {e}")
26
27    # 2. DataFrame gespeichert (bereits in df)
28
29    # 3. Überprüfen, ob die notwendigen Spalten vorhanden sind
30    required_columns = ['Bezirke', 'Straftaten_insgesamt']
31    missing_columns = [col for col in required_columns if col not in df.
32    columns]
33    if missing_columns:
34        raise ValueError(f"Die folgenden erforderlichen Spalten fehlen: {'',
35        '.join(missing_columns)}")
36    print("Alle erforderlichen Spalten sind vorhanden.")
37
38    # 4. Gesamtzahl der Straftaten für ganz Berlin finden
39    total_row = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
40    if total_row.empty:
41        raise ValueError("Die Zeile mit 'Berlin (PKS gesamt)' wurde nicht
42        gefunden.")
43    total_straftaten = total_row['Straftaten_insgesamt'].values[0]
44    print(f"Gesamtzahl der Straftaten für ganz Berlin: {total_straftaten}")
45
46    # 5. Prozentualen Anteil der einzelnen Bezirke berechnen
47    # Ausschließen der Gesamtzeile aus den Bezirken
48    bezirk_df = df[df['Bezirke'] != 'Berlin (PKS gesamt)'].copy()
49
50    # Berechnung des prozentualen Anteils
51    bezirk_df['Prozentualer_Anteil'] = (bezirk_df['Straftaten_insgesamt'] /
52    total_straftaten) * 100
53
54    # Optional: Runden auf zwei Dezimalstellen
55    bezirk_df['Prozentualer_Anteil'] = bezirk_df['Prozentualer_Anteil'].round

```

```

(2)
50
51 # 6. Ergebnis als DataFrame zurückgeben
52 result_df = bezirk_df[['Bezirke', 'Straftaten_insgesamt', '
Prozentualer_Anteil']]
53
54 print("Prozentualer Anteil der Straftaten pro Bezirk berechnet.")
55 return result_df
56
57 if __name__ == "__main__":
58     # Pfad zur Excel-Datei
59     excel_file = 'Fallzahlen.xlsx'
60
61     # Berechnung durchführen
62     try:
63         prozent_df = calculate_bezirk_straftaten_percentage(excel_file)
64         print("\nErgebnis:")
65         print(prozent_df)
66     except Exception as e:
67         print(f"Ein Fehler ist aufgetreten: {e}")

```

Listing 44: Skript Testfall 3 Prompt 3 Ausführung 3

```

1 import pandas as pd
2
3 def main():
4     # 1. Die Excel-Datei einlesen
5     try:
6         df = pd.read_excel('Fallzahlen.xlsx', sheet_name='Fallzahlen_2023')
7     except FileNotFoundError:
8         print("Die Datei 'Fallzahlen.xlsx' wurde nicht gefunden.")
9         return
10    except Exception as e:
11        print(f"Ein Fehler ist beim Einlesen der Excel-Datei aufgetreten: {e}
")
12        return
13
14    # 2. Die Tabelle als Pandas DataFrame gespeichert (bereits in 'df')
15
16    # 3. Überprüfen, ob die notwendigen Spalten vorhanden sind
17    notwendige_spalten = ['Bezirke', 'Straftaten_insgesamt']
18    fehlende_spalten = [spalte for spalte in notwendige_spalten wenn spalte
nicht in df.columns]
19    wenn fehlende_spalten:
20        print(f"Die folgenden notwendigen Spalten fehlen: {'', '.join(
fehlende_spalten)}")
21        return
22
23    # 4. Finde die Gesamtzahl der Straftaten für ganz Berlin
24    gesamt_row = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
25    wenn gesamt_row.empty:
26        print("Die Zeile mit Bezirk 'Berlin (PKS gesamt)' wurde nicht
gefunden.")

```

```

27         return
28     gesamt_straftaten = gesamt_row['Straftaten_insgesamt'].values[0]
29
30     # 5. Berechne den prozentualen Anteil der einzelnen Bezirke von ganz
    Berlin
31     # Exkludiere die Gesamtzeile aus der Berechnung
32     bezirke_df = df[df['Bezirke'] != 'Berlin (PKS gesamt)'].copy()
33     bezirke_df['Prozentualer_Anteil'] = (bezirke_df['Straftaten_insgesamt'] /
    gesamt_straftaten) * 100
34
35     # Optional: Runden auf zwei Dezimalstellen
36     bezirke_df['Prozentualer_Anteil'] = bezirke_df['Prozentualer_Anteil'].
    round(2)
37
38     # 6. Das Ergebnis als DataFrame zurückgeben
39     ergebnis_df = bezirke_df[['Bezirke', 'Straftaten_insgesamt', '
    Prozentualer_Anteil']]
40
41     # Ergebnis anzeigen
42     print(ergebnis_df)
43
44 if __name__ == "__main__":
45     main()

```

Listing 45: Skript Testfall 3 Prompt 3 Ausführung 4

```

1 import pandas as pd
2
3 def berechne_prozentanteile(dateipfad: str, sheet_name: str = '
    Fallzahlen_2023') -> pd.DataFrame:
4     """
5     Liest eine Excel-Datei ein, überprüft notwendige Spalten, berechnet den
    prozentualen Anteil
6     der Straftaten pro Bezirk und gibt das Ergebnis als DataFrame zurück.
7
8     :param dateipfad: Pfad zur Excel-Datei (z.B. 'Fallzahlen.xlsx')
9     :param sheet_name: Name des Sheets in der Excel-Datei (Standard: '
    Fallzahlen_2023')
10    :return: DataFrame mit Bezirken und deren prozentualem Anteil an den
    Straftaten
11    """
12    # 1. Excel-Datei einlesen
13    try:
14        df = pd.read_excel(dateipfad, sheet_name=sheet_name)
15        print(f"Excel-Datei '{dateipfad}' erfolgreich eingelesen.")
16    except FileNotFoundError:
17        print(f"Fehler: Die Datei '{dateipfad}' wurde nicht gefunden.")
18        return
19    except Exception as e:
20        print(f"Ein Fehler ist beim Einlesen der Excel-Datei aufgetreten: {e}
    ")
21        return
22

```

```

23 # 2. DataFrame speichern (bereits durch pd.read_excel erledigt)
24
25 # 3. Überprüfen, ob die notwendigen Spalten vorhanden sind
26 notwendige_spalten = {'Bezirke', 'Straftaten_insgesamt'}
27 vorhandene_spalten = set(df.columns)
28
29 fehlende_spalten = notwendige_spalten - vorhandene_spalten
30 if fehlende_spalten:
31     print(f"Fehler: Die folgenden notwendigen Spalten fehlen in der
32     Tabelle: {fehlende_spalten}")
33     return
34 else:
35     print("Überprüfung der notwendigen Spalten erfolgreich.")
36
37 # 4. Gesamtzahl der Straftaten für ganz Berlin finden
38 gesamt_row = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
39 if gesamt_row.empty:
40     print("Fehler: Keine Zeile mit 'Bezirke' als 'Berlin (PKS gesamt)'
41     gefunden.")
42     return
43
44 gesamt_straftaten = gesamt_row['Straftaten_insgesamt'].iloc[0]
45 print(f"Gesamtzahl der Straftaten in Berlin: {gesamt_straftaten}")
46
47 # 5. Prozentualen Anteil der einzelnen Bezirke berechnen
48 # Ausschließen der Gesamtzeile aus den Bezirken
49 bezirke_df = df[df['Bezirke'] != 'Berlin (PKS gesamt)'].copy()
50
51 # Berechnung des prozentualen Anteils
52 bezirke_df['Prozentualer_Anteil'] = (bezirke_df['Straftaten_insgesamt'] /
53     gesamt_straftaten) * 100
54
55 # Optional: Auf zwei Dezimalstellen runden
56 bezirke_df['Prozentualer_Anteil'] = bezirke_df['Prozentualer_Anteil'].
57     round(2)
58
59 # 6. Ergebnis als DataFrame zurückgeben
60 ergebnis_df = bezirke_df[['Bezirke', 'Straftaten_insgesamt', '
61     Prozentualer_Anteil']]
62
63 return ergebnis_df
64
65 # Beispielhafte Nutzung des Skripts
66 if __name__ == "__main__":
67     dateipfad = 'Fallzahlen.xlsx' # Pfad zur Excel-Datei
68     ergebnis = berechne_prozentanteile(dateipfad)
69
70     if ergebnis is not None:
71         print("\nProzentuale Anteile der Straftaten pro Bezirk:")
72         print(ergebnis)
73
74     # Optional: Ergebnis in eine neue Excel-Datei speichern

```

```

70     # ergebnis.to_excel('Prozentuale_Anteile_Straftaten.xlsx', index=
        False)

```

Listing 46: Skript Testfall 3 Prompt 3 Ausführung 5

#### 8.2.4 Testfall 4

```

1  import pandas as pd
2
3  def extract_year(sheet_name):
4      """
5      Extrahiert das Jahr aus dem Sheetnamen.
6      Annahme: Der Sheetname enthält das Jahr als vierstellige Zahl, z.B. "2020
7      ", "2021".
8      """
9      import re
10     match = re.search(r'(\d{4})', sheet_name)
11     if match:
12         return int(match.group(1))
13     else:
14         raise ValueError(f"Kein Jahr im Sheetnamen '{sheet_name}' gefunden.")
15
16 def get_total_crimes(df):
17     """
18     Extrahiert die Gesamtzahl der Straftaten für Berlin aus dem DataFrame.
19     Annahme: Die Zeile mit 'Bezirke' == 'Berlin (PKS gesamt)' enthält die
20     Gesamtzahl.
21     """
22     total_row = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
23     if total_row.empty:
24         raise ValueError("Keine Zeile mit 'Berlin (PKS gesamt)' gefunden.")
25     return total_row['Straftaten_insgesamt'].values[0]
26
27 def main():
28     # Pfad zur Excel-Datei
29     excel_file = 'Fallzahlen.xlsx'
30
31     # Lade die Excel-Datei
32     try:
33         xls = pd.ExcelFile(excel_file)
34     except FileNotFoundError:
35         print(f"Datei '{excel_file}' wurde nicht gefunden.")
36         return
37     except Exception as e:
38         print(f"Fehler beim Laden der Datei: {e}")
39         return
40
41     # Liste der Sheets
42     sheet_names = xls.sheet_names
43
44     # Liste zur Speicherung der Ergebnisse
45     data = []

```

```

44
45     for sheet in sheet_names:
46         try:
47             # Lade das Sheet als DataFrame
48             df = pd.read_excel(xls, sheet_name=sheet)
49
50             # Extrahiere das Jahr
51             year = extract_year(sheet)
52
53             # Extrahiere die Gesamtzahl der Straftaten
54             total_crimes = get_total_crimes(df)
55
56             # Speichere das Ergebnis
57             data.append({'Jahr': year, 'Straftaten_insgesamt': total_crimes})
58
59         except Exception as e:
60             print(f"Fehler beim Verarbeiten des Sheets '{sheet}': {e}")
61             continue
62
63     # Erstelle einen DataFrame aus den gesammelten Daten
64     crimes_df = pd.DataFrame(data)
65
66     # Sortiere den DataFrame nach Jahr
67     crimes_df.sort_values('Jahr', inplace=True)
68     crimes_df.reset_index(drop=True, inplace=True)
69
70     # Berechne die prozentuale Veränderung zum Vorjahr
71     crimes_df['Prozentuale_Veraenderung'] = crimes_df['Straftaten_insgesamt']
72     ].pct_change() * 100
73
74     # Optional: Runde die Prozentwerte auf zwei Dezimalstellen
75     crimes_df['Prozentuale_Veraenderung'] = crimes_df['
76     Prozentuale_Veraenderung'].round(2)
77
78     # Setze das Jahr als Index (optional)
79     crimes_df.set_index('Jahr', inplace=True)
80
81     # Anzeige des resultierenden DataFrames
82     print(crimes_df)
83
84     # Optional: Speichern des DataFrames als CSV
85     # crimes_df.to_csv('Prozentuale_Veraenderung_Straftaten_Berlin.csv')
86
87 if __name__ == "__main__":
88     main()

```

Listing 47: Skript Testfall 4 Prompt 1 Ausführung 1

```

1 import pandas as pd
2
3 def extract_berlin_total(df):
4     """
5     Extrahiert die Gesamtzahl der Straftaten für Berlin aus dem DataFrame.

```

```

6
7     Es wird entweder nach dem 'LOR-Schlüssel' 999999 oder nach dem
      Bezirksnamen
8     'Berlin (PKS gesamt)' gesucht.
9
10    Args:
11        df (pd.DataFrame): Der DataFrame des aktuellen Sheets.
12
13    Returns:
14        int oder float: Die Gesamtzahl der Straftaten in Berlin.
15    """
16    # Versuch 1: Nach LOR-Schlüssel 999999 filtern
17    berlin_row = df[df['LOR-Schlüssel'] == 999999]
18
19    if not berlin_row.empty:
20        return berlin_row['Straftaten_insgesamt'].values[0]
21
22    # Versuch 2: Nach Bezirksnamen 'Berlin (PKS gesamt)' filtern
23    berlin_row = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
24
25    if not berlin_row.empty:
26        return berlin_row['Straftaten_insgesamt'].values[0]
27
28    # Wenn nichts gefunden wurde, gebe NaN zurück
29    return float('nan')
30
31 def main():
32     # Pfad zur Excel-Datei
33     excel_file = 'Fallzahlen.xlsx'
34
35     try:
36         # Lade alle Sheets der Excel-Datei
37         xls = pd.ExcelFile(excel_file)
38     except FileNotFoundError:
39         print(f"Die Datei {excel_file} wurde nicht gefunden.")
40         return
41     except Exception as e:
42         print(f"Fehler beim Laden der Excel-Datei: {e}")
43         return
44
45     # Liste zur Speicherung der Daten
46     data = []
47
48     for sheet_name in xls.sheet_names:
49         try:
50             # Lese das aktuelle Sheet
51             df = pd.read_excel(xls, sheet_name=sheet_name)
52
53             # Extrahiere die Gesamtzahl der Straftaten für Berlin
54             total_crimes = extract_berlin_total(df)
55
56             # Versuche, das Jahr aus dem Sheet-Namen zu extrahieren

```

```

57         try:
58             year = int(sheet_name)
59         except ValueError:
60             # Wenn der Sheet-Name kein Jahr ist, überspringe dieses Sheet
61             print(f"Sheet '{sheet_name}' entspricht nicht dem erwarteten
Jahresformat und wird übersprungen.")
62             continue
63
64         # Füge die Daten zur Liste hinzu
65         data.append({'Jahr': year, 'Straftaten_insgesamt': total_crimes})
66
67     except Exception as e:
68         print(f"Fehler beim Verarbeiten des Sheets '{sheet_name}': {e}")
69         continue
70
71     # Erstelle einen DataFrame aus den gesammelten Daten
72     df_totals = pd.DataFrame(data)
73
74     if df_totals.empty:
75         print("Keine gültigen Daten gefunden.")
76         return
77
78     # Sortiere den DataFrame nach Jahr
79     df_totals.sort_values('Jahr', inplace=True)
80
81     # Setze das Jahr als Index
82     df_totals.set_index('Jahr', inplace=True)
83
84     # Berechne die prozentuale Veränderung zum Vorjahr
85     df_totals['Prozentuale_Veraenderung'] = df_totals['Straftaten_insgesamt'
].pct_change() * 100
86
87     # Optional: Runde die Prozentwerte auf zwei Dezimalstellen
88     df_totals['Prozentuale_Veraenderung'] = df_totals['
Prozentuale_Veraenderung'].round(2)
89
90     # Anzeige des resultierenden DataFrames
91     print(df_totals)
92
93     # Optional: Speichere das Ergebnis in eine neue Excel-Datei
94     output_file = 'Straftaten_Zeitreihe.xlsx'
95     try:
96         df_totals.to_excel(output_file)
97         print(f"Das Ergebnis wurde erfolgreich in '{output_file}' gespeichert
.")
98     except Exception as e:
99         print(f"Fehler beim Speichern der Ergebnisdatei: {e}")
100
101 if __name__ == "__main__":
102     main()

```

Listing 48: Skript Testfall 4 Prompt 1 Ausführung 2



```

1 import pandas as pd
2
3 # Pfad zur Excel-Datei
4 excel_datei = 'Fallzahlen.xlsx'
5
6 # Lade alle Sheets in ein Dictionary
7 alle_sheets = pd.read_excel(excel_datei, sheet_name=None)
8
9 # Liste zur Speicherung der jährlichen Straftaten
10 jährliche_straftaten = []
11
12 for sheet_name, df in alle_sheets.items():
13     try:
14         # Versuche, das Jahr aus dem Sheetnamen zu extrahieren
15         jahr = int(sheet_name)
16     except ValueError:
17         # Wenn der Sheetname kein Jahr ist, überspringe dieses Sheet
18         print(f"Sheet '{sheet_name}' übersprungen (kein gültiges Jahr).")
19         continue
20
21     # Finde die Zeile für 'Berlin (PKS gesamt)'
22     berlin_gesamt = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
23
24     if nicht berlin_gesamt.empty:
25         # Extrahiere die Gesamtzahl der Straftaten
26         gesamt_straftaten = berlin_gesamt['Straftaten_insgesamt'].values[0]
27         jährliche_straftaten.append({'Jahr': jahr, 'Straftaten_insgesamt':
gesamt_straftaten})
28     else:
29         print(f"Keine Daten für 'Berlin (PKS gesamt)' in Sheet '{sheet_name}'
gefunden.")
30
31 # Erstelle einen DataFrame aus den gesammelten Daten
32 df_jährlich = pd.DataFrame(jährliche_straftaten)
33
34 # Sortiere den DataFrame nach Jahr
35 df_jährlich = df_jährlich.sort_values('Jahr').reset_index(drop=True)
36
37 # Berechne die prozentuale Veränderung zum Vorjahr
38 df_jährlich['Veränderung_prozent'] = df_jährlich['Straftaten_insgesamt'].
pct_change() * 100
39
40 # Runde die Veränderungen auf zwei Dezimalstellen
41 df_jährlich['Veränderung_prozent'] = df_jährlich['Veränderung_prozent'].round
(2)
42
43 # Anzeige des resultierenden DataFrames
44 print(df_jährlich)

```

Listing 49: Skript Testfall 4 Prompt 1 Ausführung 3

```

1 import pandas as pd

```

```

2
3 # Pfad zur Excel-Datei
4 excel_file = 'Fallzahlen.xlsx'
5
6 # Lese alle Sheets in der Excel-Datei
7 # sheet_name=None lädt alle Sheets und gibt ein Dictionary zurück
8 sheets_dict = pd.read_excel(excel_file, sheet_name=None)
9
10 # Liste zur Speicherung der jährlichen Gesamtstraftaten
11 data = []
12
13 # Iteriere über alle Sheets
14 for sheet_name, df in sheets_dict.items():
15     # Überprüfe, ob 'LOR-Schlüssel' und 'Straftaten_insgesamt' in den Spalten
16     # vorhanden sind
17     if 'LOR-Schlüssel' in df.columns and 'Straftaten_insgesamt' in df.columns
18     :
19         # Suche die Zeile, die die Gesamtzahl für Berlin enthält
20         # Annahme: 'LOR-Schlüssel' 999999 oder 'Berlin (PKS gesamt)' reprä
21         # sentiert die Gesamtdaten
22         total_row = df[df['LOR-Schlüssel'] == 999999]
23
24         # Falls nicht gefunden, versuche mit dem Namen
25         if total_row.empty:
26             total_row = df[df['LOR-Schlüssel'].astype(str).str.contains('
27 Berlin \ (PKS gesamt\)', regex=True)]
28
29         # Falls immer noch nicht gefunden, überspringe das Sheet
30         if total_row.empty:
31             print(f"Warnung: Gesamtdaten nicht in Sheet '{sheet_name}'
32 gefunden.")
33             continue
34
35         # Extrahiere das Jahr aus dem Sheet-Namen
36         # Annahme: Der Sheet-Name enthält das Jahr, z.B. "2020", "Jahr 2020",
37         # etc.
38         # Hier wird versucht, eine vierstellige Zahl zu extrahieren
39         import re
40         match = re.search(r'\b(19|20)\d{2}\b', sheet_name)
41         if match:
42             year = int(match.group())
43         else:
44             print(f"Warnung: Jahr konnte nicht aus dem Sheet-Namen '{
45 sheet_name}' extrahiert werden. Überspringe dieses Sheet.")
46             continue
47
48         # Extrahiere die Gesamtzahl der Straftaten
49         total_crimes = total_row['Straftaten_insgesamt'].values[0]
50
51         # Füge die Daten zur Liste hinzu
52         data.append({'Jahr': year, 'Straftaten_insgesamt': total_crimes})
53     else:

```

```

47         print(f"Warnung: Erforderliche Spalten nicht in Sheet '{sheet_name}'
48             vorhanden.")
49
50 # Erstelle einen DataFrame aus den gesammelten Daten
51 df_total = pd.DataFrame(data)
52
53 # Sortiere den DataFrame nach dem Jahr
54 df_total = df_total.sort_values('Jahr').reset_index(drop=True)
55
56 # Berechne die prozentuale Veränderung zum Vorjahr
57 df_total['Prozentuale_Veraenderung_zum_Vorjahr (%)'] = df_total['
58     Straftaten_insgesamt'].pct_change() * 100
59
60 # Optional: Rundung der prozentualen Veränderung auf zwei Dezimalstellen
61 df_total['Prozentuale_Veraenderung_zum_Vorjahr (%)'] = df_total['
62     Prozentuale_Veraenderung_zum_Vorjahr (%)'].round(2)
63
64 # Setze das Jahr als Index (optional)
65 df_total.set_index('Jahr', inplace=True)
66
67 # Anzeige des Ergebnis-DataFrames
68 print(df_total)
69
70 # Optional: Speichere das Ergebnis in eine neue Excel-Datei
71 df_total.to_excel('Zeitreihe_Straftaten_Berlin.xlsx')

```

Listing 50: Skript Testfall 4 Prompt 1 Ausführung 4

```

1 import pandas as pd
2
3 # Pfad zur Excel-Datei
4 excel_file = 'Fallzahlen.xlsx'
5
6 # Excel-Datei laden
7 xls = pd.ExcelFile(excel_file)
8
9 # Liste zur Speicherung der jährlichen Gesamtdaten
10 jahresdaten = []
11
12 # Annahme: Die Sheet-Namen entsprechen den Jahreszahlen (z.B. '2018', '2019',
13     ...)
14 for sheet in xls.sheet_names:
15     # Daten des aktuellen Sheets lesen
16     df = pd.read_excel(xls, sheet_name=sheet)
17
18     # Sicherstellen, dass die 'LOR-Schlüssel' Spalte als String behandelt
19     # wird
20     df['LOR-Schlüssel'] = df['LOR-Schlüssel'].astype(str)
21
22     # Filtern nach der Zeile für ganz Berlin (angenommen '999999' ist der
23     # Schlüssel)
24     berlin_gesamt = df[df['LOR-Schlüssel'] == '999999']

```

```

23     if not berlin_gesamt.empty:
24         # Gesamtzahl der Straftaten für Berlin aus der entsprechenden Zeile
25         straftaten = berlin_gesamt['Straftaten_insgesamt'].values[0]
26     else:
27         # Wenn die 'Berlin (PKS gesamt)' Zeile nicht vorhanden ist, summiere
28         # alle Bezirke
29         straftaten = df['Straftaten_insgesamt'].sum()
30
31     # Füge die Daten zur Liste hinzu
32     jahresdaten.append({'Jahr': sheet, 'Straftaten_insgesamt': straftaten})
33
34 # Erstelle einen DataFrame aus den gesammelten Daten
35 df_gesamt = pd.DataFrame(jahresdaten)
36
37 # Versuche, die 'Jahr' Spalte in Integer zu konvertieren für die Sortierung
38 try:
39     df_gesamt['Jahr'] = df_gesamt['Jahr'].astype(int)
40     df_gesamt = df_gesamt.sort_values('Jahr')
41 except ValueError:
42     # Falls die Sheet-Namen nicht numerisch sind, sortiere alphabetisch
43     df_gesamt = df_gesamt.sort_values('Jahr')
44
45 # Setze das Jahr als Index
46 df_gesamt.set_index('Jahr', inplace=True)
47
48 # Berechne die prozentuale Veränderung zum Vorjahr
49 df_gesamt['% Veränderung Vorjahr'] = df_gesamt['Straftaten_insgesamt'].
50     pct_change() * 100
51
52 # Runde die Prozentwerte auf zwei Dezimalstellen
53 df_gesamt['% Veränderung Vorjahr'] = df_gesamt['% Veränderung Vorjahr'].round
54     (2)
55
56 # Anzeige des resultierenden DataFrames
57 print(df_gesamt)
58
59 # Optional: Speichern des DataFrames in eine neue Excel-Datei
60 df_gesamt.to_excel('Prozentuale_Veraenderung_Straftaten_Berlin.xlsx')

```

Listing 51: Skript Testfall 4 Prompt 1 Ausführung 5

```

1 import pandas as pd
2
3 # Pfad zur Excel-Datei
4 excel_file = 'Fallzahlen.xlsx'
5
6 # Lade alle Sheet-Namen
7 xls = pd.ExcelFile(excel_file)
8 sheets = xls.sheet_names
9
10 # Extrahiere das Jahr aus den Sheet-Namen und sortiere die Sheets nach Jahr
11 sheet_years = []
12 for sheet in sheets:

```

```

13     try:
14         # Erwartetes Format: 'Fallzahlen_2014', 'Fallzahlen_2015', etc.
15         year = int(sheet.split('_')[-1])
16         sheet_years.append((year, sheet))
17     except ValueError:
18         print(f"Sheet-Name '{sheet}' enthält kein gültiges Jahr und wird ü
19 bersprungen.")
20         continue
21
22 # Sortiere die Sheets nach Jahr
23 sheet_years_sorted = sorted(sheet_years, key=lambda x: x[0])
24
25 # Liste zur Speicherung der Ergebnisse
26 data = []
27
28 # Iteriere über jedes Sheet und extrahiere die benötigten Daten
29 for year, sheet in sheet_years_sorted:
30     # Lese das aktuelle Sheet
31     df = pd.read_excel(excel_file, sheet_name=sheet)
32
33     # Stelle sicher, dass die benötigten Spalten vorhanden sind
34     if 'Bezirke' not in df.columns or 'Straftaten_insgesamt' not in df.
35 columns:
36         print(f"Sheet '{sheet}' fehlt die erforderliche Spalte. Überspringe
37 dieses Sheet.")
38         continue
39
40     # Filtere die Zeile für "Berlin (PKS gesamt)"
41     row = df[df['Bezirke'] == "Berlin (PKS gesamt)"]
42
43     if not row.empty:
44         # Extrahiere den Wert und stelle sicher, dass er numerisch ist
45         value = row['Straftaten_insgesamt'].values[0]
46         if pd.notnull(value):
47             data.append({'Year': year, 'Straftaten_insgesamt': value})
48         else:
49             print(f"Der Wert für 'Straftaten_insgesamt' im Jahr {year} ist
50 NaN.")
51         else:
52             print(f"'Berlin (PKS gesamt)' nicht in Sheet '{sheet}' gefunden.")
53
54 # Erstelle ein DataFrame aus den gesammelten Daten
55 df_data = pd.DataFrame(data)
56
57 # Sortiere das DataFrame nach Jahr, um sicherzustellen, dass die Reihenfolge
58 stimmt
59 df_data = df_data.sort_values('Year').reset_index(drop=True)
60
61 # Berechne die prozentuale Veränderung zum Vorjahr
62 df_data['Prozentuale_Veraenderung'] = df_data['Straftaten_insgesamt'].
63     pct_change() * 100

```

```

59 # Optional: Runde die Prozentwerte auf zwei Dezimalstellen
60 df_data['Prozentuale_Veraenderung'] = df_data['Prozentuale_Veraenderung'].
    round(2)
61
62 # Anzeige des Ergebnisses
63 print(df_data)
64
65 # Optional: Speichere das Ergebnis in eine neue Excel-Datei oder CSV
66 # df_data.to_excel('Prozentuale_Veraenderung_Fallzahlen.xlsx', index=False)
67 # df_data.to_csv('Prozentuale_Veraenderung_Fallzahlen.csv', index=False)

```

Listing 52: Skript Testfall 4 Prompt 2 Ausführung 1

```

1 import pandas as pd
2
3 def compute_percentage_change(excel_file, target_row, target_column,
    sheet_prefix):
4     """
5     Liest eine Excel-Datei mit mehreren Sheets, extrahiert die
    Straftaten_insgesamt für
6     einen bestimmten Eintrag und berechnet die prozentuale Veränderung zum
    Vorjahr.
7
8     Parameters:
9     - excel_file: str, Pfad zur Excel-Datei.
10    - target_row: str, Der Eintrag in der 'Bezirke' Spalte, z.B. "Berlin (PKS
    gesamt)".
11    - target_column: str, Name der Spalte, deren prozentuale Veränderung
    berechnet werden soll, z.B. "Straftaten_insgesamt".
12    - sheet_prefix: str, Präfix der Sheet-Namen, z.B. "Fallzahlen_" für
    Sheets wie "Fallzahlen_2014".
13
14    Returns:
15    - pandas.DataFrame mit den Jahren und der prozentualen Veränderung.
16    """
17    try:
18        # Alle Sheets einlesen
19        sheets_dict = pd.read_excel(excel_file, sheet_name=None, engine='
    openpyxl')
20    except FileNotFoundError:
21        print(f"Die Datei {excel_file} wurde nicht gefunden.")
22        return None
23    except Exception as e:
24        print(f"Ein Fehler ist aufgetreten beim Einlesen der Datei: {e}")
25        return None
26
27    data = []
28
29    for sheet_name, df in sheets_dict.items():
30        # Überprüfen, ob der Sheet-Name dem erwarteten Muster entspricht
31        if not sheet_name.startswith(sheet_prefix):
32            print(f"Überspringe Sheet '{sheet_name}', da es nicht mit '{
    sheet_prefix}' beginnt.")

```

```

33         continue
34
35     # Extrahiere das Jahr aus dem Sheet-Namen
36     try:
37         year_str = sheet_name.replace(sheet_prefix, "")
38         year = int(year_str)
39     except ValueError:
40         print(f"Konnte das Jahr aus dem Sheet-Namen '{sheet_name}' nicht
extrahieren.")
41         continue
42
43     # Suche die Zeile mit dem gewünschten Eintrag
44     row = df[df['Bezirke'] == target_row]
45
46     if row.empty:
47         print(f"Der Eintrag '{target_row}' wurde in Sheet '{sheet_name}'
nicht gefunden.")
48         continue
49
50     # Extrahiere den Wert der Zielspalte
51     try:
52         value = row.iloc[0][target_column]
53         data.append({'Year': year, target_column: value})
54     except KeyError:
55         print(f"Die Spalte '{target_column}' wurde in Sheet '{sheet_name
}' nicht gefunden.")
56         continue
57
58     if not data:
59         print("Keine Daten gefunden, um die prozentuale Veränderung zu
berechnen.")
60         return None
61
62     # Erstelle einen DataFrame aus den gesammelten Daten
63     df_data = pd.DataFrame(data)
64
65     # Sortiere die Daten nach Jahr
66     df_data = df_data.sort_values('Year').reset_index(drop=True)
67
68     # Berechne die prozentuale Veränderung zum Vorjahr
69     df_data['Percentage_Change'] = df_data[target_column].pct_change() * 100
70
71     return df_data[['Year', 'Percentage_Change']]
72
73 if __name__ == "__main__":
74     # Parameter festlegen
75     excel_file = 'Fallzahlen.xlsx'
76     target_row = 'Berlin (PKS gesamt)'
77     target_column = 'Straftaten_insgesamt'
78     sheet_prefix = 'Fallzahlen_'
79
80     # Funktion aufrufen

```

```

81     percentage_change_df = compute_percentage_change(excel_file, target_row,
82     target_column, sheet_prefix)
83
84     if percentage_change_df is not None:
85         print("Prozentuale Veränderung der Straftaten_insgesamt zum Vorjahr:")
86     )
87     print(percentage_change_df)
88
89     # Optional: DataFrame speichern
90     # percentage_change_df.to_csv('prozentuale_veraenderung.csv', index=
91     False)

```

Listing 53: Skript Testfall 4 Prompt 2 Ausführung 2

```

1  import pandas as pd
2
3  # Pfad zur Excel-Datei
4  excel_file = 'Fallzahlen.xlsx'
5
6  # Lese alle Sheetnamen aus der Excel-Datei
7  xls = pd.ExcelFile(excel_file)
8  sheet_names = xls.sheet_names
9
10 # Initialisiere ein Dictionary, um die Daten zu speichern
11 data = {}
12
13 # Iteriere durch alle Sheets
14 for sheet in sheet_names:
15     try:
16         # Extrahiere das Jahr aus dem Sheetnamen, z.B. 'Fallzahlen_2014' ->
17         2014
18         year = int(sheet.split('_')[-1])
19     except ValueError:
20         print(f"Sheet '{sheet}' folgt nicht dem erwarteten Namensformat. Ü
21         berspringe...")
22         continue
23
24     # Lese das aktuelle Sheet
25     df = pd.read_excel(xls, sheet_name=sheet)
26
27     # Filtere die Zeile für 'Berlin (PKS gesamt)'
28     berlin_data = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
29
30     if berlin_data.empty:
31         print(f"Keine Daten für 'Berlin (PKS gesamt)' im Sheet '{sheet}'
32         gefunden.")
33         continue
34
35     # Extrahiere den Wert der Spalte 'Straftaten_insgesamt'
36     total_crimes = berlin_data['Straftaten_insgesamt'].values[0]
37
38     # Speichere die Daten im Dictionary
39     data[year] = total_crimes

```



```

37
38 # Erstelle einen DataFrame aus dem Dictionary
39 df_crimes = pd.DataFrame(list(data.items()), columns=['Jahr', '
    Straftaten_insgesamt'])
40
41 # Sortiere den DataFrame nach Jahr
42 df_crimes = df_crimes.sort_values('Jahr').reset_index(drop=True)
43
44 # Berechne die prozentuale Veränderung zum Vorjahr
45 df_crimes['Prozentuale_Veränderung (%)'] = df_crimes['Straftaten_insgesamt'].
    pct_change() * 100
46
47 # Optional: Runde die prozentuale Veränderung auf zwei Dezimalstellen
48 df_crimes['Prozentuale_Veränderung (%)'] = df_crimes['Prozentuale_Veränderung
    (%)'].round(2)
49
50 # Zeige den Ergebnis-DataFrame
51 print(df_crimes)
52
53 # Optional: Speichere den DataFrame in eine neue Excel-Datei
54 df_crimes.to_excel('Straftaten_Veränderung_Zeitreihe.xlsx', index=False)

```

Listing 54: Skript Testfall 4 Prompt 2 Ausführung 3

```

1 import pandas as pd
2
3 def analyze_fallzahlen(file_path, target_district="Berlin (PKS gesamt)",
4     column_name="Straftaten_insgesamt"):
5     """
6     Analysiert die Excel-Datei 'Fallzahlen.xlsx' und erstellt eine Zeitreihe
7     mit
8     der prozentualen Veränderung der Straftaten zum jeweiligen Vorjahr.
9
10    Parameters:
11    - file_path: Pfad zur Excel-Datei
12    - target_district: Name des Bezirks, der analysiert werden soll
13    - column_name: Name der Spalte, deren Veränderung berechnet werden soll
14
15    Returns:
16    - pandas DataFrame mit den Jahreszahlen und der prozentualen Veränderung
17    """
18    # Lade die Excel-Datei
19    try:
20        xls = pd.ExcelFile(file_path)
21    except FileNotFoundError:
22        print(f"Die Datei {file_path} wurde nicht gefunden.")
23        return
24    except Exception as e:
25        print(f"Ein Fehler ist aufgetreten beim Laden der Datei: {e}")
26        return
27
28    # Initialisiere ein Dictionary zur Speicherung der Ergebnisse

```

```

29     data = {}
30
31     # Iteriere über alle Sheets in der Excel-Datei
32     for sheet in xls.sheet_names:
33         # Extrahiere das Jahr aus dem Sheet-Namen
34         try:
35             year = int(sheet.split('_')[-1])
36         except ValueError:
37             print(f"Das Sheet '{sheet}' folgt nicht dem erwarteten
Namensformat. Überspringe dieses Sheet.")
38             continue
39
40         # Lese das aktuelle Sheet in einen DataFrame
41         try:
42             df = pd.read_excel(xls, sheet_name=sheet)
43         except Exception as e:
44             print(f"Ein Fehler ist aufgetreten beim Lesen des Sheets '{sheet
}': {e}")
45             continue
46
47         # Finde die Zeile für den Zielbezirk
48         row = df[df['Bezirke'] == target_district]
49
50         if row.empty:
51             print(f"Der Bezirk '{target_district}' wurde im Sheet '{sheet}'
nicht gefunden. Überspringe dieses Sheet.")
52             continue
53
54         # Extrahiere den Wert der gewünschten Spalte
55         try:
56             value = row[column_name].values[0]
57             # Stelle sicher, dass der Wert numerisch ist
58             value = float(value)
59         except (IndexError, ValueError) as e:
60             print(f"Fehler beim Extrahieren des Wertes in Sheet '{sheet}': {e
}")
61             continue
62
63         # Speichere das Jahr und den Wert im Dictionary
64         data[year] = value
65
66     # Erstelle einen DataFrame aus dem Dictionary
67     df_data = pd.DataFrame(list(data.items()), columns=['Year', column_name])
68
69     # Sortiere den DataFrame nach dem Jahr aufsteigend
70     df_data.sort_values('Year', inplace=True)
71     df_data.reset_index(drop=True, inplace=True)
72
73     # Berechne die prozentuale Veränderung zum Vorjahr
74     df_data['Percent Change'] = df_data[column_name].pct_change() * 100
75
76     # Optional: Runde die Prozentänderungen auf zwei Dezimalstellen

```

```

77     df_data['Percent Change'] = df_data['Percent Change'].round(2)
78
79     # Setze das Jahr als Index (optional)
80     # df_data.set_index('Year', inplace=True)
81
82     return df_data
83
84 if __name__ == "__main__":
85     # Pfad zur Excel-Datei
86     excel_file = 'Fallzahlen.xlsx'
87
88     # Führe die Analyse durch
89     result_df = analyze_fallzahlen(excel_file)
90
91     if result_df is not None:
92         # Zeige das Ergebnis an
93         print(result_df)
94
95         # Optional: Speichere das Ergebnis in einer neuen Excel- oder CSV-
96         # Datei
97         # result_df.to_excel('Straftaten_Prozentuale_Veraenderung.xlsx',
98         # index=False)
99         # oder
100        # result_df.to_csv('Straftaten_Prozentuale_Veraenderung.csv', index=
101        False)

```

Listing 55: Skript Testfall 4 Prompt 2 Ausführung 4

```

1 import pandas as pd
2 import re
3
4 def extract_year(sheet_name):
5     """
6     Extrahiert das Jahr aus dem Sheet-Namen.
7     Erwartet Format: 'Fallzahlen_YYYY'
8     """
9     match = re.search(r'Fallzahlen_(\d{4})', sheet_name)
10    if match:
11        return int(match.group(1))
12    else:
13        return None
14
15 def get_straftaten_insgesamt(excel_file, sheet_name, target_district):
16     """
17     Liest ein Sheet und extrahiert den Wert von 'Straftaten_insgesamt' für
18     den angegebenen Bezirk.
19     """
20    df = pd.read_excel(excel_file, sheet_name=sheet_name)
21
22    # Annahme: Die Bezeichnung des Bezirks ist in der Spalte 'Bezirke'
23    row = df[df['Bezirke'] == target_district]
24
25    if not row.empty:

```

```

25         return row.iloc[0]['Straftaten_insgesamt']
26     else:
27         print(f"Warnung: Bezirk '{target_district}' nicht in Sheet '{
sheet_name}' gefunden.")
28         return None
29
30 def main():
31     excel_file = 'Fallzahlen.xlsx'
32     target_district = 'Berlin (PKS gesamt)'
33
34     # Lade alle Sheet-Namen
35     xls = pd.ExcelFile(excel_file)
36     sheet_names = xls.sheet_names
37
38     # Filtere die relevanten Sheets und extrahiere die Jahre
39     pattern = re.compile(r'^Fallzahlen_(\d{4})$')
40     sheets_with_year = []
41     for sheet in sheet_names:
42         match = pattern.match(sheet)
43         if match:
44             year = int(match.group(1))
45             sheets_with_year.append((year, sheet))
46
47     if not sheets_with_year:
48         print("Keine Sheets im erwarteten Format 'Fallzahlen_YYYY' gefunden."
)
49         return
50
51     # Sortiere die Sheets nach Jahr
52     sheets_with_year.sort(key=lambda x: x[0])
53
54     # Extrahiere die 'Straftaten_insgesamt' Werte
55     data = {}
56     for year, sheet in sheets_with_year:
57         value = get_straftaten_insgesamt(excel_file, sheet, target_district)
58         if value is not None:
59             data[year] = value
60
61     # Erstelle ein DataFrame
62     df = pd.DataFrame(list(data.items()), columns=['Jahr', '
Straftaten_insgesamt'])
63     df.sort_values('Jahr', inplace=True)
64     df.reset_index(drop=True, inplace=True)
65
66     # Berechne die prozentuale Veränderung zum Vorjahr
67     df['Prozentuale_Veraenderung'] = df['Straftaten_insgesamt'].pct_change()
* 100
68
69     # Optional: Runde die Prozentwerte auf zwei Dezimalstellen
70     df['Prozentuale_Veraenderung'] = df['Prozentuale_Veraenderung'].round(2)
71
72     print(df)

```

```

73
74 if __name__ == "__main__":
75     main()

```

Listing 56: Skript Testfall 4 Prompt 2 Ausführung 5

```

1 import pandas as pd
2
3 # Schritt 1: Alle Sheets der Excel-Datei einlesen
4 excel_datei = 'Fallzahlen.xlsx'
5 xls = pd.ExcelFile(excel_datei)
6
7 # Dictionary zum Speichern der DataFrames
8 dataframes = {}
9
10 # Extrahiere die Jahreszahlen aus den Sheet-Namen und sortiere sie
11 sheet_jahre = []
12 for sheet in xls.sheet_names:
13     try:
14         # Annahme: Sheetnamen sind im Format 'Fallzahlen_Jahr', z.B. '
15         Fallzahlen_2014'
16         jahr = int(sheet.split('_')[-1])
17         sheet_jahre.append((jahr, sheet))
18     except ValueError:
19         print(f"Sheet '{sheet}' hat kein gültiges Jahresformat und wird ü
20         bersprungen.")
21
22 # Sortiere die Sheets nach Jahr
23 sheet_jahre.sort()
24
25 # Lade die DataFrames und speichere sie im Dictionary
26 for jahr, sheet in sheet_jahre:
27     df = pd.read_excel(xls, sheet_name=sheet)
28     dataframes[jahr] = df
29
30 # Schritt 2: Extrahiere 'Straftaten_insgesamt' für 'Berlin (PKS gesamt)'
31 straftaten_gesamt = {}
32
33 for jahr, df in dataframes.items():
34     # Filtern der Zeile 'Berlin (PKS gesamt)'
35     berlin_row = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
36     if not berlin_row.empty:
37         wert = berlin_row.iloc[0]['Straftaten_insgesamt']
38         straftaten_gesamt[jahr] = wert
39     else:
40         print(f"Die Zeile 'Berlin (PKS gesamt)' wurde im Jahr {jahr} nicht
41         gefunden.")
42
43 # Schritt 3: Berechne die prozentuale Veränderung zum Vorjahr
44 jahre = sorted(straftaten_gesamt.keys())
45 prozentuale_veraenderung = {}
46
47 for i in range(1, len(jahre)):

```

```

45     aktuelles_jahr = jahre[i]
46     vorheriges_jahr = jahre[i-1]
47     wert_aktuell = straftaten_gesamt[aktuelles_jahr]
48     wert_vorher = straftaten_gesamt[vorheriges_jahr]
49     veraenderung = ((wert_aktuell - wert_vorher) / wert_vorher) * 100
50     prozentuale_veraenderung[aktuelles_jahr] = veraenderung
51
52 # Schritt 4: Ergebnisse in einem neuen DataFrame speichern
53 ergebnisse = pd.DataFrame({
54     'Jahr': list(prozentuale_veraenderung.keys()),
55     'Prozentuale Veränderung (%)': list(prozentuale_veraenderung.values())
56 })
57
58 # Optional: Prozentuale Veränderung mit zwei Dezimalstellen formatieren
59 ergebnisse['Prozentuale Veränderung (%)'] = ergebnisse['Prozentuale Verä
    nderung (%)'].round(2)
60
61 # Ergebnisse anzeigen
62 print(ergebnisse)
63
64 # Optional: Ergebnisse in eine neue Excel-Datei speichern
65 ergebnisse.to_excel('Prozentuale_Veraenderung_Fallzahlen.xlsx', index=False)

```

Listing 57: Skript Testfall 4 Prompt 3 Ausführung 1

```

1 import pandas as pd
2
3 # Schritt 1: Alle Sheets der Excel-Datei einlesen
4 excel_datei = 'Fallzahlen.xlsx'
5 xls = pd.read_excel(excel_datei, sheet_name=None)
6
7 # Dictionary zum Speichern der 'Straftaten_insgesamt'-Werte pro Jahr
8 straftaten_dict = {}
9
10 # Schleife durch jedes Sheet
11 for sheet_name, df in xls.items():
12     # Extrahiere das Jahr aus dem Sheet-Namen, z.B. 'Fallzahlen_2014' -> 2014
13     try:
14         jahr = int(sheet_name.split('_')[-1])
15     except ValueError:
16         print(f"Sheet '{sheet_name}' entspricht nicht dem erwarteten Muster
    und wird übersprungen.")
17         continue
18
19     # Schritt 2: Wert der Spalte 'Straftaten_insgesamt' für 'Berlin (PKS
    gesamt)' extrahieren
20     try:
21         wert = df.loc[df['Bezirke'] == 'Berlin (PKS gesamt)', '
    Straftaten_insgesamt'].values[0]
22         straftaten_dict[jahr] = wert
23     except IndexError:
24         print(f"'Berlin (PKS gesamt)' wurde in Sheet '{sheet_name}' nicht
    gefunden.")

```

```

25     straftaten_dict[jahr] = None
26
27 # Umwandlung des Dictionaries in ein sortiertes DataFrame
28 straftaten_series = pd.Series(straftaten_dict).sort_index()
29 straftaten_df = straftaten_series.reset_index()
30 straftaten_df.columns = ['Jahr', 'Straftaten_insgesamt']
31
32 # Schritt 3: Prozentuale Veränderung zum Vorjahr berechnen
33 straftaten_df['Prozentuale_Veraenderung'] = straftaten_df['
    Straftaten_insgesamt'].pct_change() * 100
34
35 # Optional: Prozentuale Veränderung auf zwei Dezimalstellen runden
36 straftaten_df['Prozentuale_Veraenderung'] = straftaten_df['
    Prozentuale_Veraenderung'].round(2)
37
38 # Schritt 4: Ergebnisse in einem neuen DataFrame speichern
39 ergebnisse_df = straftaten_df[['Jahr', 'Prozentuale_Veraenderung']]
40
41 # Ausgabe der Ergebnisse
42 print(ergebnisse_df)
43
44 # Optional: Ergebnisse in eine neue Excel-Datei speichern
45 # ergebnisse_df.to_excel('Prozentuale_Veraenderung.xlsx', index=False)

```

Listing 58: Skript Testfall 4 Prompt 3 Ausführung 2

```

1 import pandas as pd
2
3 def main():
4     # Pfad zur Excel-Datei
5     excel_file = 'Fallzahlen.xlsx'
6
7     # Lade alle Sheet-Namen aus der Excel-Datei
8     xls = pd.ExcelFile(excel_file)
9     sheet_names = xls.sheet_names
10
11     # Dictionary zum Speichern der DataFrames
12     dataframes = {}
13
14     # Dictionary zum Speichern der 'Straftaten_insgesamt' Werte
15     strafaten_dict = {}
16
17     for sheet in sheet_names:
18         # Lese jedes Sheet in einen DataFrame
19         df = pd.read_excel(excel_file, sheet_name=sheet)
20         dataframes[sheet] = df
21
22         # Extrahiere das Jahr aus dem Sheet-Namen (angenommen, Format '
        Fallzahlen_YYYY')
23         try:
24             year = int(sheet.split('_')[-1])
25         except ValueError:
26             print(f"Sheet-Name {sheet} entspricht nicht dem erwarteten Format

```

```

    'Fallzahlen_YYYY'.")
27         continue
28
29     # Finde die Zeile für 'Berlin (PKS gesamt)'
30     berlin_row = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
31
32     if not berlin_row.empty:
33         # Extrahiere den Wert von 'Straftaten_insgesamt'
34         strafaten_value = berlin_row['Straftaten_insgesamt'].values[0]
35         strafaten_dict[year] = strafaten_value
36     else:
37         print(f"'Berlin (PKS gesamt)' nicht im Sheet {sheet} gefunden.")
38         strafaten_dict[year] = None
39
40     # Erstelle einen DataFrame aus den gesammelten Daten
41     strafaten_df = pd.DataFrame(list(strafaten_dict.items()), columns=['Jahr',
42     'Straftaten_insgesamt'])
43     strafaten_df.sort_values('Jahr', inplace=True)
44     strafaten_df.reset_index(drop=True, inplace=True)
45
46     # Berechne die prozentuale Veränderung zum Vorjahr
47     strafaten_df['Prozentuale_Veraenderung'] = strafaten_df['
48     Straftaten_insgesamt'].pct_change() * 100
49
50     # Optional: Runde die Veränderung auf 2 Dezimalstellen
51     strafaten_df['Prozentuale_Veraenderung'] = strafaten_df['
52     Prozentuale_Veraenderung'].round(2)
53
54     # Anzeige der Ergebnisse
55     print(strafaten_df)
56
57     # Optional: Speichern der Ergebnisse in eine neue Excel-Datei
58     output_file = 'Straftaten_Veraenderung.xlsx'
59     strafaten_df.to_excel(output_file, index=False)
60     print(f"Ergebnisse wurden in '{output_file}' gespeichert.")
61
62 if __name__ == "__main__":
63     main()

```

Listing 59: Skript Testfall 4 Prompt 3 Ausführung 3

```

1 import pandas as pd
2
3 # 1. Lesen Sie alle Sheets der Excel-Datei ein und speichern Sie jedes Sheet
4   in einem separaten DataFrame
5 excel_file = 'Fallzahlen.xlsx'
6 xls = pd.read_excel(excel_file, sheet_name=None) # Liest alle Sheets als ein
7   Dictionary
8
9 # Dictionary zur Speicherung der 'Straftaten_insgesamt' Werte pro Jahr
10 strafaten_dict = {}
11
12 for sheet_name, df in xls.items():

```



```

11 # Extrahiere das Jahr aus dem Sheet-Namen, z.B. 'Fallzahlen_2014' -> 2014
12 try:
13     year = int(sheet_name.split('_')[-1])
14 except ValueError:
15     print(f"Sheet-Name '{sheet_name}' entspricht nicht dem erwarteten
16     Muster 'Fallzahlen_Jahr'. Übersprungen.")
17     continue
18
19 # 2. Extrahiere den Wert der Spalte 'Straftaten_insgesamt' für die Zeile
20 # 'Berlin (PKS gesamt)'
21 # Überprüfen Sie, ob die Spaltennamen korrekt sind. Möglicherweise müssen
22 # Sie 'Bezirke' anpassen.
23 # Hier wird angenommen, dass die Spalte mit Bezirken 'Bezirke' heißt
24 berlin_row = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
25
26 if not berlin_row.empty:
27     straf_value = berlin_row['Straftaten_insgesamt'].values[0]
28     straftaten_dict[year] = straf_value
29 else:
30     print(f"'Berlin (PKS gesamt)' nicht in Sheet '{sheet_name}' gefunden.
31     ")
32
33 # 3. Berechnen Sie die prozentuale Veränderung zum jeweiligen Vorjahr
34 # Sortieren der Jahre
35 sorted_years = sorted(straftaten_dict.keys())
36 percentage_changes = []
37
38 for i in range(1, len(sorted_years)):
39     current_year = sorted_years[i]
40     previous_year = sorted_years[i - 1]
41     current_value = straftaten_dict[current_year]
42     previous_value = straftaten_dict[previous_year]
43
44     # Prozentuale Veränderung berechnen
45     if previous_value != 0:
46         percent_change = ((current_value - previous_value) / previous_value)
47         * 100
48     else:
49         percent_change = None # Vermeiden von Division durch Null
50
51     percentage_changes.append({
52         'Jahr': current_year,
53         'Prozentuale Veränderung': percent_change
54     })
55
56 # 4. Speichern Sie die Ergebnisse in einem neuen DataFrame
57 result_df = pd.DataFrame(percentage_changes)
58
59 # Optional: Hinzufügen des ersten Jahres mit einer NaN für die Veränderung
60 result_df = pd.concat([
61     pd.DataFrame({'Jahr': [sorted_years[0]], 'Prozentuale Veränderung': [None]}),

```

```

57     result_df
58 ], ignore_index=True)
59
60 # Anzeige des Ergebnisses
61 print(result_df)
62
63 # Optional: Speichern des Ergebnisses in eine neue Excel-Datei
64 result_df.to_excel('Prozentuale_Veraenderung_Straftaten.xlsx', index=False)

```

Listing 60: Skript Testfall 4 Prompt 3 Ausführung 4

```

1  import pandas as pd
2
3  # 1. Lesen Sie alle Sheets der Excel-Datei ein
4  excel_file = 'Fallzahlen.xlsx'
5  all_sheets = pd.read_excel(excel_file, sheet_name=None)
6
7  # Liste zur Speicherung der Ergebnisse
8  daten_liste = []
9
10 # 2. Extrahieren Sie den Wert der Spalte 'Straftaten_insgesamt' für 'Berlin (
    PKS gesamt)'
11 for sheet_name, df in all_sheets.items():
12     # Extrahieren Sie das Jahr aus dem Sheet-Namen, z.B. 'Fallzahlen_2014' ->
        2014
13     try:
14         jahr = int(sheet_name.split('_')[-1])
15     except ValueError:
16         print(f"Das Sheet '{sheet_name}' hat keinen gültigen Jahresnamen und
            wird übersprungen.")
17         continue
18
19     # Suche nach der Zeile 'Berlin (PKS gesamt)'
20     berlin_data = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
21
22     if berlin_data.empty:
23         print(f"'Berlin (PKS gesamt)' wurde im Sheet '{sheet_name}' nicht
            gefunden.")
24         continue
25
26     # Extrahiere den Wert der Spalte 'Straftaten_insgesamt'
27     strftaten_insgesamt = berlin_data['Straftaten_insgesamt'].values[0]
28
29     daten_liste.append({'Jahr': jahr, 'Straftaten_insgesamt':
        strftaten_insgesamt})
30
31 # 3. Erstellen Sie ein DataFrame aus den gesammelten Daten
32 daten_df = pd.DataFrame(daten_liste)
33
34 # Sortieren Sie das DataFrame nach Jahr aufsteigend
35 daten_df = daten_df.sort_values('Jahr').reset_index(drop=True)
36
37 # 4. Berechnen Sie die prozentuale Veränderung zum Vorjahr

```

```

38 daten_df['Prozentuale_Veraenderung'] = daten_df['Straftaten_insgesamt'].
    pct_change() * 100
39
40 # Optional: Runden Sie die Prozentsätze auf zwei Dezimalstellen
41 daten_df['Prozentuale_Veraenderung'] = daten_df['Prozentuale_Veraenderung'].
    round(2)
42
43 # 5. Erstellen Sie das Endergebnis DataFrame
44 ergebnis_df = daten_df[['Jahr', 'Prozentuale_Veraenderung']]
45
46 # Optional: Benennen Sie die Spalten auf Deutsch um
47 ergebnis_df.columns = ['Jahr', 'Prozentuale Veränderung (%)']
48
49 # 6. Anzeigen des Ergebnisses
50 print(ergebnis_df)
51
52 # 7. Speichern Sie das Ergebnis in einer neuen Excel-Datei (optional)
53 ergebnis_df.to_excel('Prozentuale_Veraenderung_Fallzahlen.xlsx', index=False)

```

Listing 61: Skript Testfall 4 Prompt 3 Ausführung 5

### 8.2.5 Manuelle Skripte

```

1 import pandas as pd
2
3 file = 'Fallzahlen.xlsx'
4 sheet = 'Fallzahlen_2023'
5
6 # Read the excel file
7 try:
8     df = pd.read_excel(file, sheet_name=sheet)
9 except FileNotFoundError:
10     print(f"Datei '{file}' wurde nicht gefunden.")
11     exit(1)
12
13 # Check for required column
14 required_columns = ['LOR-Schlüssel', 'Straftaten_insgesamt']
15 missing_columns = [col for col in required_columns if col not in df.columns]
16 if missing_columns:
17     raise ValueError(f"Fehlende Spalten in der Excel-Datei: {missing_columns}")
18
19 # Extract non sort rows and save them for later
20 extract_keys = [999900, 999999]
21 extract_rows = df[df['LOR-Schlüssel'].isin(extract_keys)]
22 df = df[~df['LOR-Schlüssel'].isin(extract_keys)]
23
24 # Sort dataframe and concat with extracted rows
25 sorted_df = df.sort_values(by='Straftaten_insgesamt', ascending=False)
26 sorted_df = pd.concat([sorted_df, extract_rows], ignore_index=True)
27
28 print(sorted_df)

```

Listing 62: Manuelles Skript Testfall 1

```
1 import pandas as pd
2
3 # Read excel file
4 file = 'Fallzahlen.xlsx'
5 sheets = pd.read_excel(file, sheet_name=None)
6
7 # Join all sheets
8 joined_df = pd.concat(sheets.values(), ignore_index=True)
9 grouped_df = joined_df.groupby(['LOR-Schlüssel', 'Bezirke'], as_index=False).
    sum()
10
11 # Exclude not to sort rows
12 exclude_keys = [999900, 999999]
13 to_sort_df = grouped_df[~grouped_df['LOR-Schlüssel'].isin(exclude_keys)]
14 excluded_df = grouped_df[grouped_df['LOR-Schlüssel'].isin(exclude_keys)]
15
16 # Sort DF
17 sorted_df = to_sort_df.sort_values(by='Straftaten_insgesamt', ascending=False)
18
19 # Concat excluded rows
20 final_df = pd.concat([sorted_df, excluded_df], ignore_index=True)
21
22 print(final_df)
```

Listing 63: Manuelles Skript Testfall 2

```
1 import pandas as pd
2
3 # Read Excel file
4 df = pd.read_excel('Fallzahlen.xlsx', sheet_name='Fallzahlen_2023', usecols=[
    'Bezirke', 'Straftaten_insgesamt'])
5
6 # Extract total and save for later
7 total = df['Straftaten_insgesamt'].iloc[-1]
8 df = df.iloc[:-1]
9
10 # Calculate percentage
11 df['% Anteil'] = (df['Straftaten_insgesamt'] / total) * 100
12 df['% Anteil'] = df['% Anteil'].round(2)
13
14 # Add total row
15 df_total = pd.DataFrame({
16     'Bezirke': ['Berlin (PKS gesamt)'],
17     'Straftaten_insgesamt': [total],
18     '% Anteil': [100]
19 })
20 df = pd.concat([df, df_total], ignore_index=True)
21
```

```
22 print(df)
```

Listing 64: Manuelles Skript Testfall 3

```
1 import pandas as pd
2
3 # Read Excel
4 df = pd.read_excel('Fallzahlen.xlsx', sheet_name=None, usecols=['Bezirke', '
    Straftaten_insgesamt'])
5
6 gesamtwerte = []
7 # Loop all sheets
8 for sheet_name, data in df.items():
9     # Get year from sheet name
10    year = sheet_name.split('_')[-1]
11    # Get total
12    total = data["Straftaten_insgesamt"].iloc[-1]
13    # Append to list
14    gesamtwerte.append((year, total))
15
16 # Create DataFrame and add % change
17 df_total = pd.DataFrame(gesamtwerte, columns=['Jahr', 'Straftaten_insgesamt'
    ])
18 df_total["% Entwicklung"] = df_total['Straftaten_insgesamt'].pct_change() *
    100
19 df_total["% Entwicklung"] = df_total["% Entwicklung"].round(2)
20
21 print(df_total)
```

Listing 65: Manuelles Skript Testfall 4

### 8.2.6 Komplette Antworten der AI Requests

```
1 Natürlich! Hier ist ein Python-Skript, das die gewünschten Schritte ausführt:
2
3 1. **Installiere die notwendigen Bibliotheken** (falls noch nicht geschehen):
4     '''bash
5     pip install pandas openpyxl
6     '''
7
8 2. **Python-Skript**:
9
10    '''python
11    import pandas as pd
12
13    # Pfad zur Excel-Datei
14    datei_pfad = 'Fallzahlen.xlsx'
15
16    try:
17        # Einlesen des spezifischen Sheets 'Fallzahlen_2023'
18        df = pd.read_excel(datei_pfad, sheet_name='Fallzahlen_2023')
19
```

```

20     # Anzeigen der ersten paar Zeilen zur Überprüfung
21     print("Original DataFrame:")
22     print(df.head())
23
24     # Sortieren nach der Spalte 'Straftaten_insgesamt' in absteigender
    Reihenfolge
25     df_sortiert = df.sort_values(by='Straftaten_insgesamt', ascending=
    False)
26
27     # Zurücksetzen des Index (optional)
28     df_sortiert.reset_index(drop=True, inplace=True)
29
30     # Anzeige der sortierten DataFrame
31     print("\nSortierter DataFrame nach 'Straftaten_insgesamt':")
32     print(df_sortiert)
33
34     # Optional: Speichern des sortierten DataFrames in eine neue Excel-
    Datei
35     sortierte_datei_pfad = 'Fallzahlen_sortiert.xlsx'
36     df_sortiert.to_excel(sortierte_datei_pfad, index=False)
37     print(f"\nDer sortierte DataFrame wurde in '{sortierte_datei_pfad}'
    gespeichert.")
38
39 except FileNotFoundError:
40     print(f"Die Datei '{datei_pfad}' wurde nicht gefunden.")
41 except ValueError as ve:
42     print(f"Fehler beim Einlesen des Sheets: {ve}")
43 except Exception as e:
44     print(f"Ein unerwarteter Fehler ist aufgetreten: {e}")
45     '''
46
47 3. **Erklärung der Schritte**:
48
49 - **Importieren der Bibliotheken**:
50     '''python
51     import pandas as pd
52     '''
53     'pandas' ist eine leistungsstarke Bibliothek zur Datenanalyse in Python.
54
55 - **Einlesen der Excel-Datei**:
56     '''python
57     df = pd.read_excel(datei_pfad, sheet_name='Fallzahlen_2023')
58     '''
59     Hier wird das spezifische Sheet 'Fallzahlen_2023' aus der Excel-Datei
    eingelesen.
60
61 - **Sortieren des DataFrames**:
62     '''python
63     df_sortiert = df.sort_values(by='Straftaten_insgesamt', ascending=False)
64     '''
65     Der DataFrame wird basierend auf der Spalte 'Straftaten_insgesamt' in
    absteigender Reihenfolge sortiert.

```

```

66
67 - **Zurücksetzen des Index**:
68     '''python
69     df_sortiert.reset_index(drop=True, inplace=True)
70     '''
71     Dies entfernt den alten Index und setzt einen neuen, fortlaufenden Index
72     .
73
74 - **Speichern der sortierten Daten** (optional):
75     '''python
76     df_sortiert.to_excel(sortierte_datei_pfad, index=False)
77     '''
78     Der sortierte DataFrame wird in eine neue Excel-Datei '
79     Fallzahlen_sortiert.xlsx' gespeichert.
80
81 - **Fehlerbehandlung**:
82     Der 'try-except' Block stellt sicher, dass mögliche Fehler beim Einlesen
83     der Datei oder beim Verarbeiten des Sheets abgefangen und verständliche
84     Fehlermeldungen ausgegeben werden.
85
86
87
88 4. **Ausführen des Skripts**:
89     Speichere das obige Skript in einer '.py' Datei, zum Beispiel '
90     sortiere_fallzahlen.py', und führe es in deiner Python-Umgebung aus:
91     '''bash
92     python sortiere_fallzahlen.py
93     '''
94
95 Dieses Skript liest die Daten aus der Excel-Datei, sortiert sie nach der
96 Gesamtzahl der Straftaten und speichert das Ergebnis in einem neuen
97 DataFrame. Optional kannst du das sortierte Ergebnis auch in eine neue
98 Excel-Datei exportieren.

```

Listing 66: AI Response Testfall 1 Prompt 1 Ausführung 1

```

1 Natürlich, ich helfe dir gerne dabei, ein Python-Skript zu erstellen, das
2 deine Excel-Datei einliest, die Daten nach der Spalte **"
3 Straftaten_insgesamt"** sortiert und in einem DataFrame speichert. Wir
4 werden dafür die **pandas** Bibliothek verwenden, die sich hervorragend fü
5 r die Datenanalyse eignet. Falls du **pandas** und **openpyxl** noch nicht
6 installiert hast, kannst du dies mit 'pip' tun.
7
8 ### Schritt 1: Installation der benötigten Bibliotheken
9
10 Öffne dein Terminal oder die Eingabeaufforderung und führe die folgenden
11 Befehle aus:
12
13     '''bash
14     pip install pandas openpyxl
15     '''
16
17 ### Schritt 2: Python-Skript erstellen
18
19 Erstelle eine neue Python-Datei (z.B. 'sortiere_fallzahlen.py') und füge den

```

```

    folgenden Code ein:
14
15 '''python
16 import pandas as pd
17
18 # Pfad zur Excel-Datei
19 excel_datei = 'Fallzahlen.xlsx'
20
21 # Name des Sheets
22 sheet_name = 'Fallzahlen_2023'
23
24 # Einlesen der Excel-Datei
25 df = pd.read_excel(excel_datei, sheet_name=sheet_name)
26
27 # Anzeigen der ersten paar Zeilen (optional)
28 print("Unsortierte Daten:")
29 print(df.head())
30
31 # Sortieren nach 'Straftaten_insgesamt' in absteigender Reihenfolge
32 df_sortiert = df.sort_values(by='Straftaten_insgesamt', ascending=False)
33
34 # Anzeigen der sortierten Daten (optional)
35 print("\nSortierte Daten nach 'Straftaten_insgesamt':")
36 print(df_sortiert.head())
37
38 # Optional: Speichern des sortierten DataFrames in einer neuen Excel-Datei
39 df_sortiert.to_excel('Fallzahlen_sortiert.xlsx', index=False)
40
41 print("\nDie sortierten Daten wurden in 'Fallzahlen_sortiert.xlsx'
    gespeichert.")
42 '''
43
44 ### Erklärung des Codes
45
46 1. **Import der pandas-Bibliothek:**
47     '''python
48     import pandas as pd
49     '''
50     Dadurch kannst du die pandas-Funktionen verwenden, um mit den Daten zu
    arbeiten.
51
52 2. **Pfad und Sheet-Name festlegen:**
53     '''python
54     excel_datei = 'Fallzahlen.xlsx'
55     sheet_name = 'Fallzahlen_2023'
56     '''
57     Stelle sicher, dass sich die Excel-Datei im gleichen Verzeichnis wie dein
    Python-Skript befindet. Andernfalls musst du den vollständigen Pfad
    angeben.
58
59 3. **Einlesen der Excel-Datei:**
60     '''python

```



```

61 df = pd.read_excel(excel_datei, sheet_name=sheet_name)
62 '''
63 Dies liest das angegebene Sheet in einen pandas DataFrame ein.
64
65 4. **Anzeigen der unsortierten Daten (optional):**
66 '''python
67 print("Unsortierte Daten:")
68 print(df.head())
69 '''
70 Dies hilft dir, zu überprüfen, ob die Daten korrekt eingelesen wurden.
71
72 5. **Sortieren der Daten:**
73 '''python
74 df_sortiert = df.sort_values(by='Straftaten_insgesamt', ascending=False)
75 '''
76 Hier werden die Daten nach der Spalte "Straftaten_insgesamt" in
77 absteigender Reihenfolge sortiert. Wenn du in aufsteigender Reihenfolge
78 sortieren möchtest, setze 'ascending=True'.
79
80 6. **Anzeigen der sortierten Daten (optional):**
81 '''python
82 print("\nSortierte Daten nach 'Straftaten_insgesamt':")
83 print(df_sortiert.head())
84 '''
85 Dadurch kannst du die oberen Einträge der sortierten Daten überprüfen.
86
87 7. **Speichern der sortierten Daten in einer neuen Excel-Datei (optional):**
88 '''python
89 df_sortiert.to_excel('Fallzahlen_sortiert.xlsx', index=False)
90 '''
91 Dies speichert den sortierten DataFrame in einer neuen Excel-Datei namens
92 "Fallzahlen_sortiert.xlsx". Der Parameter 'index=False' verhindert,
93 dass der DataFrame-Index in die Excel-Datei geschrieben wird.
94
95 8. **Abschlussnachricht:**
96 '''python
97 print("\nDie sortierten Daten wurden in 'Fallzahlen_sortiert.xlsx'
98 gespeichert.")
99 '''
100
101 ### Ausführen des Skripts
102
103 Nachdem du das Skript erstellt hast, kannst du es ausführen, indem du im
104 Terminal oder der Eingabeaufforderung in das Verzeichnis navigierst, in
105 dem sich dein Skript befindet, und den folgenden Befehl eingibst:
106
107 '''bash
108 python sortiere_fallzahlen.py
109 '''
110
111 Nach der Ausführung solltest du eine neue Excel-Datei namens "Fallzahlen_sortiert.xlsx" in deinem Verzeichnis finden, die die

```

```

    sortierten Daten enthält.
105
106 ### Zusätzliche Hinweise
107
108 - **Überprüfen der Spaltennamen:** Stelle sicher, dass die Spaltennamen in
    deiner Excel-Datei genau mit den im Skript verwendeten Namen ü
    bereinstimmen. Groß- und Kleinschreibung sowie Leerzeichen müssen exakt
    passen. Andernfalls erhältst du möglicherweise einen Fehler.
109
110 - **Fehlerbehandlung:** Für eine robustere Anwendung kannst du
    Fehlerbehandlung hinzufügen, um Probleme wie das Nichtfinden der Datei
    oder fehlende Spalten abzufangen.
111
112 Hier ist ein erweitertes Beispiel mit einfacher Fehlerbehandlung:
113
114 ```python
115 import pandas as pd
116 import sys
117
118 excel_datei = 'Fallzahlen.xlsx'
119 sheet_name = 'Fallzahlen_2023'
120
121 try:
122     df = pd.read_excel(excel_datei, sheet_name=sheet_name)
123 except FileNotFoundError:
124     print(f"Fehler: Die Datei '{excel_datei}' wurde nicht gefunden.")
125     sys.exit(1)
126 except ValueError:
127     print(f"Fehler: Das Sheet '{sheet_name}' wurde nicht gefunden.")
128     sys.exit(1)
129
130 if 'Straftaten_insgesamt' not in df.columns:
131     print("Fehler: Die Spalte 'Straftaten_insgesamt' wurde in der Excel-Datei
    nicht gefunden.")
132     sys.exit(1)
133
134 df_sortiert = df.sort_values(by='Straftaten_insgesamt', ascending=False)
135 df_sortiert.to_excel('Fallzahlen_sortiert.xlsx', index=False)
136 print("Die sortierten Daten wurden erfolgreich gespeichert in '
    Fallzahlen_sortiert.xlsx'.")
137 ```
138
139 Dieses erweiterte Skript überprüft, ob die Datei und das angegebene Sheet
    existieren und ob die benötigte Spalte vorhanden ist, bevor es die
    Sortierung durchführt.
140
141 Ich hoffe, das hilft dir weiter! Wenn du weitere Fragen hast, stehe ich gerne
    zur Verfügung.

```

Listing 67: AI Response Testfall 1 Prompt 1 Ausführung 2

```

1 Gerne erstelle ich dir ein vollständiges Python-Skript, das die gewünschten
    Schritte ausführt. Wir verwenden dazu die Bibliothek 'pandas', die sich

```

```

hervorragend für die Arbeit mit Excel-Dateien und Datenmanipulation eignet
. Stelle sicher, dass du 'pandas' und 'openpyxl' (für das Lesen von '.xlsx
'-Dateien) installiert hast. Wenn nicht, kannst du sie mit 'pip'
installieren:
2
3 '''bash
4 pip install pandas openpyxl
5 '''
6
7 Hier ist das vollständige Skript:
8
9 '''python
10 import pandas as pd
11
12 # Pfad zur Excel-Datei
13 excel_datei = 'Fallzahlen.xlsx'
14
15 # Name des Sheets
16 sheet_name = 'Fallzahlen_2023'
17
18 try:
19     # Einlesen der Excel-Datei
20     df = pd.read_excel(excel_datei, sheet_name=sheet_name)
21     print("Excel-Datei erfolgreich eingelesen.")
22
23     # Überprüfen, ob die erfordernte Spalte existiert
24     if 'Straftaten_insgesamt' not in df.columns:
25         raise ValueError("Die Spalte 'Straftaten_insgesamt' wurde im
DataFrame nicht gefunden.")
26
27     # Sortieren des DataFrames nach 'Straftaten_insgesamt' in absteigender
Reihenfolge
28     df_sorted = df.sort_values(by='Straftaten_insgesamt', ascending=False).
reset_index(drop=True)
29     print("DataFrame erfolgreich nach 'Straftaten_insgesamt' sortiert.")
30
31     # Optional: Anzeige der sortierten Daten
32     print(df_sorted)
33
34     # Optional: Speichern des sortierten DataFrames in eine neue Excel-Datei
35     # df_sorted.to_excel('Fallzahlen_sorted.xlsx', index=False)
36     # print("Sortierte Daten wurden in 'Fallzahlen_sorted.xlsx' gespeichert
.")
37
38 except FileNotFoundError:
39     print(f"Die Datei '{excel_datei}' wurde nicht gefunden.")
40 except ValueError as ve:
41     print(f"Fehler: {ve}")
42 except Exception as e:
43     print(f"Ein unerwarteter Fehler ist aufgetreten: {e}")
44 '''
45

```

```

46 ### Erklärung des Skripts
47
48 1. **Importieren der benötigten Bibliothek:**
49     '''python
50     import pandas as pd
51     '''
52
53 2. **Definieren des Pfads zur Excel-Datei und des Sheet-Namens:**
54     '''python
55     excel_datei = 'Fallzahlen.xlsx'
56     sheet_name = 'Fallzahlen_2023'
57     '''
58
59 3. **Einlesen der Excel-Datei:**
60     '''python
61     df = pd.read_excel(excel_datei, sheet_name=sheet_name)
62     '''
63     - 'pd.read_excel' liest die Excel-Datei ein.
64     - 'sheet_name' gibt das spezifische Sheet an, das eingelesen werden soll.
65
66 4. **Überprüfen, ob die erforderliche Spalte existiert:**
67     '''python
68     if 'Straftaten_insgesamt' not in df.columns:
69         raise ValueError("Die Spalte 'Straftaten_insgesamt' wurde im DataFrame
70             nicht gefunden.")
71     '''
72     - Dies stellt sicher, dass die Spalte, nach der sortiert werden soll, auch
73       tatsächlich vorhanden ist.
74
75 5. **Sortieren des DataFrames:**
76     '''python
77     df_sorted = df.sort_values(by='Straftaten_insgesamt', ascending=False).
78         reset_index(drop=True)
79     '''
80     - 'sort_values' sortiert den DataFrame nach der angegebenen Spalte.
81     - 'ascending=False' sortiert in absteigender Reihenfolge. Setze dies auf '
82       True', wenn du eine aufsteigende Sortierung möchtest.
83     - 'reset_index(drop=True)' setzt die Indizes neu und entfernt die alten
84       Indizes.
85
86 6. **Optional: Anzeige der sortierten Daten:**
87     '''python
88     print(df_sorted)
89     '''
90
91 7. **Optional: Speichern des sortierten DataFrames in eine neue Excel-Datei
92     : **
93     '''python
94     df_sorted.to_excel('Fallzahlen_sorted.xlsx', index=False)
95     '''
96     - Dies speichert den sortierten DataFrame in einer neuen Datei namens '
97       Fallzahlen_sorted.xlsx'.

```

```

91 - Der 'index=False' Parameter stellt sicher, dass die Indizes nicht als
    separate Spalte gespeichert werden.
92
93 8. **Fehlerbehandlung:**
94 - Das Skript enthält grundlegende Fehlerbehandlungen für häufige Probleme
    wie fehlende Dateien oder Spalten.
95
96 ### Ausführung des Skripts
97
98 1. **Speichere das Skript in einer Datei**, zum Beispiel 'sortiere_fallzahlen
    .py'.
99
100 2. **Stelle sicher, dass sich die Datei 'Fallzahlen.xlsx'** im selben
    Verzeichnis wie das Skript befindet oder passe den Pfad entsprechend an.
101
102 3. **Führe das Skript aus:**
103     '''bash
104     python sortiere_fallzahlen.py
105     '''
106
107 Nach der Ausführung solltest du die sortierten Daten im Terminal sehen. Wenn
    du die Zeile zum Speichern des sortierten DataFrames aktivierst (indem du
    die Kommentarzeichen entfernst), wird auch eine neue Excel-Datei mit den
    sortierten Daten erstellt.
108
109 ### Anpassungen
110
111 - **Sortierreihenfolge ändern:**
112 - Für eine aufsteigende Sortierung ändere 'ascending=False' zu 'ascending=
    True'.
113
114 - **Sortieren nach mehreren Spalten:**
115     '''python
116     df_sorted = df.sort_values(by=['Straftaten_insgesamt', 'Raub'], ascending=[
117     False, True])
118     '''
119 - Dies sortiert zuerst nach 'Straftaten_insgesamt' absteigend und dann nach
    'Raub' aufsteigend.
120
121 Wenn du weitere Anpassungen oder zusätzliche Funktionen benötigst, lass es
    mich wissen!

```

Listing 68: AI Response Testfall 1 Prompt 1 Ausführung 3

```

1 Natürlich, ich helfe dir gerne dabei, ein Python-Skript zu erstellen, das
    deine Excel-Datei einliest, die Daten nach der Spalte '
    Straftaten_insgesamt' sortiert und in einem DataFrame speichert. Dafür
    verwenden wir die Bibliothek 'pandas', die sich hervorragend für solche
    Aufgaben eignet.
2
3 Hier ist ein vollständiges Skript, das du verwenden kannst:
4
5 '''python

```

```

6 import pandas as pd
7
8 # Pfad zur Excel-Datei
9 excel_datei = 'Fallzahlen.xlsx'
10
11 # Name des Sheets
12 sheet_name = 'Fallzahlen_2023'
13
14 # Excel-Datei einlesen
15 df = pd.read_excel(excel_datei, sheet_name=sheet_name)
16
17 # Daten nach 'Straftaten_insgesamt' in absteigender Reihenfolge sortieren
18 df_sortiert = df.sort_values(by='Straftaten_insgesamt', ascending=False)
19
20 # Optional: Den sortierten DataFrame anzeigen
21 print(df_sortiert)
22
23 # Optional: Den sortierten DataFrame in eine neue Excel-Datei speichern
24 df_sortiert.to_excel('Fallzahlen_sortiert.xlsx', index=False)
25 '''
26
27 ### Schritt-für-Schritt-Erklärung:
28
29 1. **Importieren der pandas-Bibliothek:**
30     '''python
31     import pandas as pd
32     '''
33     'pandas' ist eine leistungsstarke Bibliothek für Datenmanipulation und -
34     analyse.
35
36 2. **Definieren des Pfads zur Excel-Datei und des Sheet-Namens:**
37     '''python
38     excel_datei = 'Fallzahlen.xlsx'
39     sheet_name = 'Fallzahlen_2023'
40     '''
41
42 3. **Einlesen der Excel-Datei:**
43     '''python
44     df = pd.read_excel(excel_datei, sheet_name=sheet_name)
45     '''
46     Dies liest die Daten aus dem angegebenen Sheet in einen DataFrame 'df' ein
47     .
48
49 4. **Sortieren der Daten nach 'Straftaten_insgesamt':**
50     '''python
51     df_sortiert = df.sort_values(by='Straftaten_insgesamt', ascending=False)
52     '''
53     Hier sortierst du die Daten basierend auf der Spalte 'Straftaten_insgesamt'
54     in absteigender Reihenfolge. Wenn du in aufsteigender Reihenfolge
55     sortieren möchtest, setze 'ascending=True'.
56
57 5. **Anzeigen des sortierten DataFrames (optional):**

```

```

54     '''python
55     print(df_sortiert)
56     '''
57     Dies gibt den sortierten DataFrame im Terminal oder der Konsole aus.
58
59 6. **Speichern des sortierten DataFrames in eine neue Excel-Datei (optional)
   :**
60     '''python
61     df_sortiert.to_excel('Fallzahlen_sortiert.xlsx', index=False)
62     '''
63     Dies speichert die sortierten Daten in einer neuen Excel-Datei namens '
        Fallzahlen_sortiert.xlsx'. Der Parameter 'index=False' sorgt dafür, dass
        der Index nicht mitgespeichert wird.
64
65 ### Zusätzliche Hinweise:
66
67 - **Installation von pandas:**
68     Falls du 'pandas' noch nicht installiert hast, kannst du es mit folgendem
        Befehl installieren:
69     '''bash
70     pip install pandas
71     '''
72
73 - **Installation von openpyxl (falls benötigt):**
74     Um Excel-Dateien mit 'pandas' zu lesen und zu schreiben, benötigst du
        eventuell das Paket 'openpyxl'. Installiere es mit:
75     '''bash
76     pip install openpyxl
77     '''
78
79 - **Fehlerbehebung:**
80     - **Dateipfad:** Stelle sicher, dass sich die Datei 'Fallzahlen.xlsx' im
        gleichen Verzeichnis wie dein Skript befindet oder gib den vollständigen
        Pfad zur Datei an.
81     - **Spaltenname:** Achte darauf, dass der Spaltenname 'Straftaten_insgesamt'
        genau so geschrieben ist wie in deiner Excel-Datei, einschließlich Groß-
        und Kleinschreibung.
82
83 Mit diesem Skript solltest du problemlos deine Excel-Daten einlesen,
        sortieren und weiterverarbeiten können. Wenn du weitere Anpassungen benö-
        tigtst oder Fragen hast, lasse es mich wissen!

```

Listing 69: AI Response Testfall 1 Prompt 1 Ausführung 4

```

1 Hier ist ein Python-Skript, das die Excel-Datei 'Fallzahlen.xlsx' einliest,
   die Daten nach der Spalte 'Straftaten_insgesamt' sortiert und in einem
   DataFrame speichert. Anschließend wird das sortierte DataFrame angezeigt.
2
3 **Voraussetzungen:**
4 - Installiere die benötigten Bibliotheken, falls noch nicht geschehen. Du
   kannst dies mit 'pip' tun:
5
6     '''bash

```

```

7 pip install pandas openpyxl
8 '''
9
10 **Python-Skript:**
11
12 '''python
13 import pandas as pd
14
15 # Pfad zur Excel-Datei
16 dateipfad = 'Fallzahlen.xlsx'
17
18 # Name des Sheets
19 sheet_name = 'Fallzahlen_2023'
20
21 # Einlesen der Excel-Datei
22 try:
23     df = pd.read_excel(dateipfad, sheet_name=sheet_name)
24     print("Daten erfolgreich eingelesen.")
25 except FileNotFoundError:
26     print(f"Die Datei {dateipfad} wurde nicht gefunden.")
27     exit()
28 except ValueError:
29     print(f"Das Sheet '{sheet_name}' existiert nicht in der Datei.")
30     exit()
31
32 # Überprüfen der Spaltennamen (optional)
33 print("Verfügbare Spalten:")
34 print(df.columns.tolist())
35
36 # Sortieren nach 'Straftaten_insgesamt' in absteigender Reihenfolge
37 # Falls aufsteigend gewünscht ist, setze ascending=True
38 try:
39     df_sortiert = df.sort_values(by='Straftaten_insgesamt', ascending=False)
40     print("Daten erfolgreich sortiert.")
41 except KeyError:
42     print("Die Spalte 'Straftaten_insgesamt' wurde nicht gefunden.")
43     exit()
44
45 # Zurücksetzen des Indexes (optional)
46 df_sortiert.reset_index(drop=True, inplace=True)
47
48 # Anzeigen des sortierten DataFrames
49 print("Sortiertes DataFrame:")
50 print(df_sortiert)
51 '''
52
53 **Erklärung der Schritte:**
54
55 1. **Importieren der Bibliothek:**
56     - 'pandas' wird für die Datenmanipulation verwendet.
57
58 2. **Definieren des Pfads und Sheet-Namens:**

```



```

59 - Stelle sicher, dass der Dateipfad korrekt ist. Wenn sich die Excel-Datei
    im selben Verzeichnis wie das Skript befindet, reicht der Dateiname.
60
61 3. **Einlesen der Excel-Datei:**
62 - 'pd.read_excel' liest die Excel-Datei ein. Es wird das spezifische Sheet
    'Fallzahlen_2023' ausgewählt.
63 - Fehlerbehandlung ist integriert, um sicherzustellen, dass die Datei und
    das Sheet existieren.
64
65 4. **Überprüfen der Spaltennamen:**
66 - Dies ist optional, hilft aber sicherzustellen, dass die gewünschte
    Spalte vorhanden ist.
67
68 5. **Sortieren der Daten:**
69 - Die Daten werden nach der Spalte 'Straftaten_insgesamt' sortiert.
    Standardmäßig geschieht dies in absteigender Reihenfolge ('ascending=False
    '), sodass die Bezirke mit den meisten Straftaten oben stehen.
70 - Falls eine aufsteigende Sortierung gewünscht ist, setze 'ascending=True
    '.
71
72 6. **Zurücksetzen des Indexes:**
73 - Dies ist optional, sorgt aber dafür, dass der Index des sortierten
    DataFrames konsistent ist.
74
75 7. **Anzeigen des sortierten DataFrames:**
76 - Das sortierte DataFrame wird ausgegeben, sodass du die sortierten Daten
    sehen kannst.
77
78 **Hinweis:**
79 Stelle sicher, dass die Spaltennamen exakt mit den in deiner Excel-Datei ü
    bereinstimmen, einschließlich Groß- und Kleinschreibung und etwaiger
    Leerzeichen. Falls die Spalte 'Straftaten_insgesamt' anders benannt ist,
    passe den 'by'-Parameter entsprechend an.

```

Listing 70: AI Response Testfall 1 Prompt 1 Ausführung 5

```

1 Hier ist ein Python-Skript, das die gewünschten Schritte ausführt:
2
3 1. **Installiere die notwendigen Bibliotheken** (falls noch nicht installiert
    ):
4     '''bash
5     pip install pandas openpyxl
6     '''
7
8 2. **Python-Skript**:
9
10    '''python
11    import pandas as pd
12
13    # Pfad zur Excel-Datei
14    excel_datei = 'Fallzahlen.xlsx'
15
16    # Name des Sheets

```

```

17 sheet_name = 'Fallzahlen_2023'
18
19 # LOR-Schlüssel, die ausgeschlossen werden sollen
20 ausschluss_schluessel = [999900, 999999]
21
22 try:
23     # Excel-Datei einlesen
24     df = pd.read_excel(excel_datei, sheet_name=sheet_name)
25
26     # Anzeige der ersten paar Zeilen zur Überprüfung (optional)
27     print("Originaldaten:")
28     print(df.head())
29
30     # Filter anwenden, um die auszuschließenden LOR-Schlüssel zu entfernen
31     df_filtered = df[~df['LOR-Schlüssel'].isin(ausschluss_schluessel)]
32
33     # Anzeige der gefilterten Daten zur Überprüfung (optional)
34     print("\nGefilterte Daten:")
35     print(df_filtered.head())
36
37     # Sortieren nach 'Straftaten_insgesamt' absteigend (größte zuerst)
38     df_sorted = df_filtered.sort_values(by='Straftaten_insgesamt',
39                                         ascending=False)
40
41     # Ergebnis anzeigen (optional)
42     print("\nSortierte Daten:")
43     print(df_sorted)
44
45     # Optional: Das sortierte DataFrame speichern
46     # df_sorted.to_excel('Fallzahlen_sortiert.xlsx', index=False)
47
48 except FileNotFoundError:
49     print(f"Die Datei {excel_datei} wurde nicht gefunden.")
50 except Exception as e:
51     print(f"Ein Fehler ist aufgetreten: {e}")
52
53 3. **Erklärung der Schritte**:
54
55 - **Importieren der Bibliotheken**:
56     '''python
57     import pandas as pd
58     '''
59     Pandas ist eine leistungsstarke Bibliothek zur Datenmanipulation und -
60     analyse.
61
62 - **Einlesen der Excel-Datei**:
63     '''python
64     df = pd.read_excel(excel_datei, sheet_name=sheet_name)
65     '''
66     Liest das spezifizierte Sheet aus der Excel-Datei in ein DataFrame ein.

```

```

67 - **Filtern der unerwünschten Zeilen**:
68     '''python
69     df_filtered = df[~df['LOR-Schlüssel'].isin(ausschluss_schluesel)]
70     '''
71     Entfernt die Zeilen, bei denen der 'LOR-Schlüssel' entweder 999900 oder
72     999999 ist.
73
74 - **Sortieren der Daten**:
75     '''python
76     df_sorted = df_filtered.sort_values(by='Straftaten_insgesamt', ascending
77     =False)
78     '''
79     Sortiert das gefilterte DataFrame basierend auf der Spalte '
80     Straftaten_insgesamt' in absteigender Reihenfolge. Wenn eine aufsteigende
81     Sortierung gewünscht ist, setze 'ascending=True'.
82
83 - **Optionales Speichern der sortierten Daten**:
84     '''python
85     df_sorted.to_excel('Fallzahlen_sortiert.xlsx', index=False)
86     '''
87     Speichert das sortierte DataFrame in einer neuen Excel-Datei. Diese
88     Zeile ist auskommentiert, kann aber aktiviert werden, wenn das Ergebnis
89     gespeichert werden soll.
90
91 4. **Ausführen des Skripts**:
92     Speichere das Skript in einer '.py'-Datei, z.B. 'sortiere_fallzahlen.py',
93     und führe es mit Python aus:
94     '''bash
95     python sortiere_fallzahlen.py
96     '''
97
98 Dieses Skript liest die Daten aus der Excel-Datei, filtert die nicht
99 relevanten Bezirke heraus, sortiert die verbleibenden Daten nach der
100 Gesamtzahl der Straftaten und gibt das sortierte DataFrame aus. Optional
101 kannst du das sortierte Ergebnis in einer neuen Excel-Datei speichern,
102 indem du die entsprechende Zeile entkommentierst.

```

Listing 71: AI Response Testfall 1 Prompt 2 Ausführung 1

```

1 Natürlich! Hier ist ein Python-Skript, das die beschriebenen Anforderungen
2 erfüllt. Das Skript liest die Excel-Datei ein, filtert die unerwünschten
3 Zeilen heraus, sortiert die verbleibenden Daten nach der Spalte '
4 Straftaten_insgesamt' und speichert das Ergebnis in einem Pandas
5 DataFrame.
6
7 ### Voraussetzungen
8
9 Stellen Sie sicher, dass Sie die benötigten Python-Bibliotheken installiert
10 haben. Sie benötigen 'pandas' und 'openpyxl'. Falls diese noch nicht
11 installiert sind, können Sie sie mit 'pip' installieren:
12
13 '''bash
14 pip install pandas openpyxl

```

```

9  '''
10
11  ### Python-Skript
12
13  '''python
14  import pandas as pd
15
16  # Pfad zur Excel-Datei
17  excel_datei = 'Fallzahlen.xlsx'
18
19  # Name des Sheets
20  sheet_name = 'Fallzahlen_2023'
21
22  # LOR-Schlüssel, die ausgeschlossen werden sollen
23  ausgeschlossene_lor_schluessel = [999900, 999999]
24
25  # Einlesen der Excel-Datei
26  try:
27      df = pd.read_excel(excel_datei, sheet_name=sheet_name, engine='openpyxl')
28      print("Excel-Datei erfolgreich eingelesen.")
29  except FileNotFoundError:
30      print(f"Die Datei {excel_datei} wurde nicht gefunden.")
31      exit(1)
32  except Exception as e:
33      print(f"Ein Fehler ist aufgetreten beim Einlesen der Excel-Datei: {e}")
34      exit(1)
35
36  # Filtern der Zeilen, die nicht zu Bezirken gehören
37  df_filtered = df[~df['LOR-Schlüssel'].isin(ausgeschlossene_lor_schluessel)]
38
39  # Sortieren nach 'Straftaten_insgesamt' in absteigender Reihenfolge
40  df_sorted = df_filtered.sort_values(by='Straftaten_insgesamt', ascending=
    False).reset_index(drop=True)
41
42  # Anzeige des sortierten DataFrames
43  print("Sortierte Daten nach 'Straftaten_insgesamt':")
44  print(df_sorted)
45
46  # Optional: Speichern des sortierten DataFrames in eine neue Excel-Datei
47  # df_sorted.to_excel('Fallzahlen_sortiert.xlsx', index=False)
48  '''
49
50  ### Erklärung der Schritte
51
52  1. **Importieren der Bibliotheken:**
53      '''python
54      import pandas as pd
55      '''
56
57  2. **Definieren der Variablen:**
58      - 'excel_datei': Der Name der Excel-Datei.
59      - 'sheet_name': Der Name des Sheets, das eingelesen werden soll.

```

```

60 - 'ausgeschlossene_lor_schluessel': Eine Liste der 'LOR-Schlüssel', die
    ausgeschlossen werden sollen.
61
62 3. **Einlesen der Excel-Datei:**
63     '''python
64     df = pd.read_excel(excel_datei, sheet_name=sheet_name, engine='openpyxl')
65     '''
66 - Verwendet die 'read_excel'-Funktion von Pandas mit dem angegebenen Sheet
    .
67 - Der 'engine='openpyxl' Parameter stellt sicher, dass die '.xlsx' Datei
    korrekt gelesen wird.
68
69 4. **Filtern der unerwünschten Zeilen:**
70     '''python
71     df_filtered = df[~df['LOR-Schlüssel'].isin(ausgeschlossene_lor_schluessel)
72     ]
73     '''
74 - 'isin' prüft, ob der 'LOR-Schlüssel' in der Ausschlussliste ist.
75 - Das '~' Symbol negiert die Bedingung, sodass nur die Zeilen behalten
    werden, die **nicht** in der Ausschlussliste sind.
76
77 5. **Sortieren der Daten:**
78     '''python
79     df_sorted = df_filtered.sort_values(by='Straftaten_insgesamt', ascending=
80     False).reset_index(drop=True)
81     '''
82 - Sortiert die gefilterten Daten nach 'Straftaten_insgesamt' in
    absteigender Reihenfolge ('ascending=False').
83 - 'reset_index(drop=True)' setzt den Index des DataFrames zurück, sodass
    er fortlaufend von 0 beginnt.
84
85 6. **Anzeige des Ergebnisses:**
86     '''python
87     print(df_sorted)
88     '''
89 - Gibt den sortierten DataFrame in der Konsole aus.
90
91 7. **Optional: Speichern in eine neue Excel-Datei:**
92     '''python
93     df_sorted.to_excel('Fallzahlen_sortiert.xlsx', index=False)
94     '''
95 - Speichert den sortierten DataFrame in eine neue Excel-Datei namens '
    Fallzahlen_sortiert.xlsx'.
96 - 'index=False' vermeidet, dass der DataFrame-Index in der Excel-Datei
    gespeichert wird.
97
98 ### Ausführen des Skripts
99
100 Speichern Sie das Skript in einer Datei, z.B. 'sortiere_fallzahlen.py', und f
    ühren Sie es in Ihrer Python-Umgebung aus:
101
102     '''bash

```

```

101 python sortiere_fallzahlen.py
102 '''
103
104 Nach der Ausführung sollten die sortierten Daten im Terminal angezeigt werden
    . Wenn Sie die Option zum Speichern aktiviert haben, finden Sie die
    sortierten Daten auch in der neuen Excel-Datei 'Fallzahlen_sortiert.xlsx'.
105
106 ### Beispielausgabe
107
108 Nach dem Ausführen des Skripts sollte eine sortierte Tabelle angezeigt werden
    , zum Beispiel:
109
110 '''
111 Sortierte Daten nach 'Straftaten_insgesamt':
112
113   LOR-Schlüssel      Bezirke  Straftaten_insgesamt  Raub  Straß
114   enraub ... Sachbeschädigung_Graffiti  Rauschgiftdelikte  Kieztaten
115 0           10000           Mitte           83611    884
116   521 ...           976           3306    19212
117 1           20000 Friedrichshain-Kreuzberg           55838    923
118   606 ...           801           4455    12724
119 2           80000           Neukölln           45313    508
120   294 ...           379           1869    10253
121 ...
122 '''
123
124 *(Die tatsächliche Ausgabe hängt von den Daten in Ihrer Excel-Datei ab.)*
125
126 ---
127
128 Falls Sie weitere Anpassungen oder Funktionen benötigen, lassen Sie es mich
    wissen!

```

Listing 72: AI Response Testfall 1 Prompt 2 Ausführung 2

```

1 Natürlich! Hier ist ein Python-Skript, das die beschriebenen Anforderungen
    erfüllt. Das Skript verwendet die 'pandas'-Bibliothek, um die Excel-Datei
    einzulesen, die Daten zu filtern, zu sortieren und in einem DataFrame zu
    speichern.
2
3 Stellen Sie sicher, dass Sie die 'pandas'-Bibliothek installiert haben. Wenn
    nicht, können Sie sie mit 'pip' installieren:
4
5 '''bash
6 pip install pandas
7 '''
8
9 Hier ist das vollständige Skript:
10
11 '''python
12 import pandas as pd
13
14 # Pfad zur Excel-Datei
15 dateipfad = 'Fallzahlen.xlsx'

```

```

16
17 # Name des Sheets
18 sheet_name = 'Fallzahlen_2023'
19
20 # Einlesen der Excel-Datei
21 df = pd.read_excel(dateipfad, sheet_name=sheet_name)
22
23 # Anzeigen der ersten Zeilen zur Überprüfung (optional)
24 #print(df.head())
25
26 # Filtere die Bezirke heraus, indem die unerwünschten LOR-Schlüssel
    ausgeschlossen werden
27 ausgeschlossene_lor = [9999900, 9999999]
28 bezirke_df = df[~df['LOR-Schlüssel'].isin(ausgeschlossene_lor)]
29
30 # Sortiere die Bezirke nach 'Straftaten_insgesamt' in absteigender
    Reihenfolge
31 bezirke_sortiert = bezirke_df.sort_values(by='Straftaten_insgesamt',
    ascending=False)
32
33 # Optional: Zurücksetzen des Indexes nach dem Sortieren
34 bezirke_sortiert.reset_index(drop=True, inplace=True)
35
36 # Anzeigen des sortierten DataFrames (optional)
37 print(bezirke_sortiert)
38
39 # Wenn Sie den sortierten DataFrame weiter verwenden möchten, steht er als '
    bezirke_sortiert' zur Verfügung
40 '''
41
42 ### Erklärung der Schritte:
43
44 1. **Importieren der Bibliothek:**
45     '''python
46     import pandas as pd
47     '''
48     'pandas' ist eine leistungsstarke Bibliothek für Datenmanipulation und -
        analyse.
49
50 2. **Definieren des Pfads und des Sheets:**
51     '''python
52     dateipfad = 'Fallzahlen.xlsx'
53     sheet_name = 'Fallzahlen_2023'
54     '''
55     Stellen Sie sicher, dass sich die Excel-Datei im gleichen Verzeichnis wie
        das Skript befindet oder geben Sie den vollständigen Pfad an.
56
57 3. **Einlesen der Excel-Datei:**
58     '''python
59     df = pd.read_excel(dateipfad, sheet_name=sheet_name)
60     '''
61     Dies liest das spezifizierte Sheet in einen Pandas DataFrame ein.

```

```

62
63 4. **Filtern der Bezirke:**
64     '''python
65     ausgeschlossen_lor = [999900, 999999]
66     bezirke_df = df[~df['LOR-Schlüssel'].isin(ausgeschlossene_lor)]
67     '''
68     - 'ausgeschlossene_lor' enthält die LOR-Schlüssel, die ausgeschlossen
        werden sollen.
69     - '~' ist der logische NOT-Operator in Pandas.
70     - 'isin' prüft, ob Werte in der Spalte 'LOR-Schlüssel' in der Liste '
        ausgeschlossene_lor' enthalten sind.
71     - Damit bleiben nur die Bezirke übrig.
72
73 5. **Sortieren nach 'Straftaten_insgesamt':**
74     '''python
75     bezirke_sortiert = bezirke_df.sort_values(by='Straftaten_insgesamt',
        ascending=False)
76     '''
77     Sortiert die Bezirke in absteigender Reihenfolge basierend auf der Anzahl
        der Straftaten insgesamt. Falls Sie aufsteigend sortieren möchten, setzen
        Sie 'ascending=True'.
78
79 6. **Zurücksetzen des Indexes (optional):**
80     '''python
81     bezirke_sortiert.reset_index(drop=True, inplace=True)
82     '''
83     Dies setzt den Index neu, sodass er von 0 beginnend fortlaufend ist. 'drop
        =True' verhindert, dass der alte Index als Spalte hinzugefügt wird.
84
85 7. **Anzeigen des sortierten DataFrames (optional):**
86     '''python
87     print(bezirke_sortiert)
88     '''
89     Dies gibt den sortierten DataFrame im Terminal oder der Konsole aus.
90
91 ### Weiterverwendung
92
93 Der sortierte DataFrame 'bezirke_sortiert' steht Ihnen für weitere Analysen
        oder Operationen zur Verfügung. Zum Beispiel können Sie ihn in eine neue
        Excel-Datei speichern:
94
95     '''python
96     bezirke_sortiert.to_excel('Sortierte_Fallzahlen.xlsx', index=False)
97     '''
98
99 ### Zusätzliche Hinweise
100
101 - **Encoding und weitere Optionen:** Falls Ihre Excel-Datei spezielle
        Encoding-Anforderungen hat oder zusätzliche Optionen benötigt, können Sie
        diese im 'read_excel'-Befehl angeben.
102
103 - **Fehlerbehandlung:** Es kann sinnvoll sein, Fehlerbehandlungen hinzuzufü

```



```

    gen, um sicherzustellen, dass die Datei existiert und das Sheet korrekt
    benannt ist.
104
105 Ich hoffe, das hilft Ihnen weiter! Wenn Sie weitere Fragen haben oder
    Anpassungen benötigen, lassen Sie es mich wissen.

```

Listing 73: AI Response Testfall 1 Prompt 2 Ausführung 3

```

1 Natürlich! Hier ist ein Python-Skript, das die beschriebenen Anforderungen
    erfüllt. Dieses Skript verwendet die 'pandas'-Bibliothek, um die Excel-
    Datei zu lesen, die unerwünschten Zeilen zu entfernen, die verbleibenden
    Bezirke nach der Spalte 'Straftaten_insgesamt' zu sortieren und das
    Ergebnis in einem DataFrame zu speichern.
2
3 ### Voraussetzungen
4
5 Stelle sicher, dass du die benötigten Bibliotheken installiert hast. Falls
    nicht, kannst du sie mit 'pip' installieren:
6
7 ```bash
8 pip install pandas openpyxl
9 ```
10
11 ### Python-Skript
12
13 ```python
14 import pandas as pd
15
16 # Pfad zur Excel-Datei
17 excel_datei = 'Fallzahlen.xlsx'
18
19 # Namen des Sheets
20 sheet_name = 'Fallzahlen_2023'
21
22 # LOR-Schlüssel der Zeilen, die ausgeschlossen werden sollen
23 ausschluss_lor_schluessel = [999900, 999999]
24
25 # Excel-Datei einlesen
26 df = pd.read_excel(excel_datei, sheet_name=sheet_name)
27
28 # Anzeigen der ursprünglichen Daten (optional)
29 print("Ursprüngliche Daten:")
30 print(df)
31
32 # Filtern der Bezirke, indem die ausgeschlossenen LOR-Schlüssel entfernt
    werden
33 df_gefiltert = df[~df['LOR-Schlüssel'].isin(ausschluss_lor_schluessel)]
34
35 # Anzeigen der gefilterten Daten (optional)
36 print("\nGefilterte Daten (ohne nicht zuzuordnende Bezirke):")
37 print(df_gefiltert)
38
39 # Sortieren nach 'Straftaten_insgesamt' in absteigender Reihenfolge

```

```

40 df_sortiert = df_gefiltert.sort_values(by='Straftaten_insgesamt', ascending=
    False)
41
42 # Zurücksetzen des Index (optional, für sauberere Anzeige)
43 df_sortiert.reset_index(drop=True, inplace=True)
44
45 # Anzeigen der sortierten Daten (optional)
46 print("\nSortierte Daten nach 'Straftaten_insgesamt':")
47 print(df_sortiert)
48
49 # Optional: Speichern des sortierten DataFrames in eine neue Excel-Datei
50 # df_sortiert.to_excel('Sortierte_Fallzahlen.xlsx', index=False)
51 '''
52
53 ### Erklärung der Schritte
54
55 1. **Importieren der Bibliothek:**
56     '''python
57     import pandas as pd
58     '''
59     Wir verwenden 'pandas' für die Datenmanipulation und das Einlesen der
60     Excel-Datei.
61
62 2. **Festlegen des Dateipfades und des Sheet-Namens:**
63     '''python
64     excel_datei = 'Fallzahlen.xlsx'
65     sheet_name = 'Fallzahlen_2023'
66     '''
67
68 3. **Definieren der auszuschließenden LOR-Schlüssel:**
69     '''python
70     ausschluss_lor_schluessel = [999900, 999999]
71     '''
72     Diese Schlüssel entsprechen den Zeilen, die nicht zu Bezirken gehören und
73     daher bei der Sortierung ignoriert werden sollen.
74
75 4. **Einlesen der Excel-Datei:**
76     '''python
77     df = pd.read_excel(excel_datei, sheet_name=sheet_name)
78     '''
79     Hiermit wird das Sheet 'Fallzahlen_2023' aus der Datei 'Fallzahlen.xlsx
80     ' in ein DataFrame 'df' geladen.
81
82 5. **Filtern der unerwünschten Zeilen:**
83     '''python
84     df_gefiltert = df[~df['LOR-Schlüssel'].isin(ausschluss_lor_schluessel)]
85     '''
86     - 'df['LOR-Schlüssel'].isin(ausschluss_lor_schluessel)' überprüft, welche
87       Zeilen einen der auszuschließenden LOR-Schlüssel haben.
88     - Das '~' negiert das Ergebnis, sodass nur die Zeilen behalten werden, die
89       **nicht** in der Ausschlussliste sind.

```

```

86 6. **Sortieren nach 'Straftaten_insgesamt':**
87   '''python
88   df_sortiert = df_gefiltert.sort_values(by='Straftaten_insgesamt',
89   ascending=False)
90   '''
91   Das DataFrame 'df_gefiltert' wird basierend auf der Spalte '
92   Straftaten_insgesamt' in absteigender Reihenfolge sortiert.
93
94 7. **Zurücksetzen des Index (optional):**
95   '''python
96   df_sortiert.reset_index(drop=True, inplace=True)
97   '''
98   Dies ist hilfreich, um einen sauberen, fortlaufenden Index im sortierten
99   DataFrame zu haben.
100
101 8. **Speichern des sortierten DataFrames (optional):**
102   '''python
103   # df_sortiert.to_excel('Sortierte_Fallzahlen.xlsx', index=False)
104   '''
105   Falls du das Ergebnis in eine neue Excel-Datei speichern möchtest, kannst
106   du diese Zeile entkommentieren.
107
108 ### Beispielausgabe
109
110 Das Skript enthält 'print'-Anweisungen, die die Daten in den verschiedenen
111 Schritten anzeigen. Dies ist nützlich für die Überprüfung, kann aber bei
112 großen Datensätzen entfernt oder angepasst werden.
113
114 '''plaintext
115 Ursprüngliche Daten:
116   LOR-Schlüssel           Bezirke  Straftaten_insgesamt  Raub  Stra
117   ßenraub  Körperverletzungen_insgesamt  ...  Branddelikte_insgesamt
118   Brandstiftung  Sachbeschädigung_insgesamt  Sachbeschädigung_Graffiti
119   Rauschgiftdelikte  Kieztaten
120
121 0           10000           Mitte           83611    884
122   521           8144  ...           420
123   137           6172           976
124   3306           19212
125
126 1           20000  Friedrichshain-Kreuzberg           55838    923
127   606           5502  ...           298
128   85           4906           801
129   4455           12724
130
131 ...           ...           ...           ...    ...
132   ...           ...           ...           ...    ...
133   ...           ...           ...           ...    ...
134   ...           ...
135
136 Gefilterte Daten (ohne nicht zuzuordnende Bezirke):
137   LOR-Schlüssel           Bezirke  Straftaten_insgesamt  Raub  Stra
138   ßenraub  Körperverletzungen_insgesamt  ...  Branddelikte_insgesamt
139   Brandstiftung  Sachbeschädigung_insgesamt  Sachbeschädigung_Graffiti
140   Rauschgiftdelikte  Kieztaten

```

```

117 0          10000          Mitte          83611    884
      521          8144    ...          420
      137          6172          976
      3306          19212
118 1          20000    Friedrichshain-Kreuzberg          55838    923
      606          5502    ...          298
      85          4906          801
      4455          12724
119 ...          ...          ...          ...
      ...          ...          ...
      ...          ...
120
121 Sortierte Daten nach 'Straftaten_insgesamt':
122 LOR-Schlüssel          Bezirke    Straftaten_insgesamt    Raub    Stra
    ßenraub    Körperverletzungen_insgesamt    ...    Branddelikte_insgesamt
    Brandstiftung    Sachbeschädigung_insgesamt    Sachbeschädigung_Graffiti
    Rauschgiftdelikte    Kieztaten
123 0          10000          Mitte          83611    884
      521          8144    ...          420
      137          6172          976
      3306          19212
124 1          20000    Friedrichshain-Kreuzberg          55838    923
      606          5502    ...          298
      85          4906          801
      4455          12724
125 ...          ...          ...          ...
      ...          ...          ...
      ...          ...
126 ' ' '
127
128 ### Anpassungen nach Bedarf
129
130 - **Dateipfad anpassen:** Stelle sicher, dass der Pfad zur Excel-Datei
      korrekt ist. Wenn sich die Datei in einem anderen Verzeichnis befindet,
      gib den vollständigen Pfad an, z.B. './daten/Fallzahlen.xlsx'.
131
132 - **Erweiterte Filterbedingungen:** Falls du weitere Filterkriterien hinzufü
      gen möchtest, kannst du die Filterlogik entsprechend erweitern.
133
134 - **Speichern in anderen Formaten:** Neben Excel kannst du das DataFrame auch
      in andere Formate wie CSV speichern, indem du 'df_sortiert.to_csv('
      Sortierte_Fallzahlen.csv', index=False)'' verwendest.
135
136 ### Zusammenfassung
137
138 Dieses Skript liest die Excel-Datei ein, entfernt die Zeilen, die keine
      Bezirke darstellen, sortiert die verbleibenden Bezirke nach der Gesamtzahl
      der Straftaten und speichert das Ergebnis in einem Pandas DataFrame. Du
      kannst das sortierte DataFrame weiter analysieren, visualisieren oder in
      andere Formate exportieren, je nach deinen Bedürfnissen.

```

```
139
140 Wenn du weitere Anpassungen oder Erweiterungen benötigst, lass es mich wissen
    !
```

Listing 74: AI Response Testfall 1 Prompt 2 Ausführung 4

```
1 Natürlich! Hier ist ein Python-Skript, das die beschriebenen Anforderungen
  erfüllt. Es verwendet die 'pandas'-Bibliothek, um die Excel-Datei
  einzulesen, die unerwünschten Zeilen zu filtern und die verbleibenden
  Daten nach der Spalte 'Straftaten_insgesamt' zu sortieren.
2
3 ### Schritt-für-Schritt Anleitung:
4
5 1. **Installation der notwendigen Bibliotheken**:
6 Stelle sicher, dass 'pandas' und 'openpyxl' (für das Einlesen von '.xlsx'-
  Dateien) installiert sind. Wenn nicht, kannst du sie mit 'pip'
  installieren:
7
8   ```bash
9   pip install pandas openpyxl
10  ```
11
12 2. **Python-Skript erstellen**:
13
14   ```python
15   import pandas as pd
16
17   # Pfad zur Excel-Datei
18   datei_pfad = 'Fallzahlen.xlsx'
19
20   # Namen des Sheets
21   sheet_name = 'Fallzahlen_2023'
22
23   # Einlesen der Excel-Datei
24   try:
25       df = pd.read_excel(datei_pfad, sheet_name=sheet_name)
26       print("Excel-Datei erfolgreich eingelesen.")
27   except FileNotFoundError:
28       print(f"Die Datei {datei_pfad} wurde nicht gefunden.")
29       exit(1)
30   except Exception as e:
31       print(f"Ein Fehler ist aufgetreten beim Einlesen der Datei: {e}")
32       exit(1)
33
34   # Anzeigen der ersten paar Zeilen, um sicherzustellen, dass die Daten
35   korrekt eingelesen wurden
36   print("\nErste 5 Zeilen der Originaldaten:")
37   print(df.head())
38
39   # Filterkriterien: Ausschließen der LOR-Schlüssel 999900 und 999999
40   ausschluss_schluesel = [999900, 999999]
41   df_filtered = df[~df['LOR-Schlüssel'].isin(ausschluss_schluesel)]
```

```

42 print(f"\nAnzahl der Zeilen nach dem Ausschluss: {len(df_filtered)}")
43
44 # Sortieren nach 'Straftaten_insgesamt' in absteigender Reihenfolge
45 df_sorted = df_filtered.sort_values(by='Straftaten_insgesamt', ascending=
46 False).reset_index(drop=True)
47
48 print("\nErste 5 Zeilen der sortierten Daten:")
49 print(df_sorted.head())
50
51 # Optional: Speichern des sortierten DataFrames in eine neue Excel-Datei
52 ausgabe_pfad = 'Fallzahlen_sortiert.xlsx'
53 try:
54     df_sorted.to_excel(ausgabe_pfad, index=False)
55     print(f"\nSortierte Daten wurden erfolgreich in '{ausgabe_pfad}'
56 gespeichert.")
57 except Exception as e:
58     print(f"Ein Fehler ist aufgetreten beim Speichern der Datei: {e}")
59
60 ### Erklärung des Skripts:
61
62 - **Importieren der Bibliothek**:
63     '''python
64     import pandas as pd
65     '''
66     'pandas' ist eine leistungsstarke Bibliothek für Datenmanipulation und -
67     analyse.
68
69 - **Einlesen der Excel-Datei**:
70     '''python
71     df = pd.read_excel(datei_pfad, sheet_name=sheet_name)
72     '''
73     Diese Zeile liest das angegebene Sheet der Excel-Datei in einen DataFrame '
74     df' ein.
75
76 - **Filtern der unerwünschten Zeilen**:
77     '''python
78     df_filtered = df[~df['LOR-Schlüssel'].isin(ausschluss_schluesssel)]
79     '''
80     Hier werden alle Zeilen ausgeschlossen, deren 'LOR-Schlüssel' entweder
81     '999900' oder '999999' ist. Das '~' Symbol negiert die Bedingung.
82
83 - **Sortieren der Daten**:
84     '''python
85     df_sorted = df_filtered.sort_values(by='Straftaten_insgesamt', ascending=
86 False).reset_index(drop=True)
87     '''
88     Diese Zeile sortiert die gefilterten Daten nach der Spalte '
89     Straftaten_insgesamt' in absteigender Reihenfolge. Das 'reset_index(drop=
90 True)' stellt sicher, dass die Indizes neu gesetzt werden und die alten
91 Indizes verworfen werden.

```

```

85 - **Speichern der sortierten Daten (optional)**:
86     '''python
87     df_sorted.to_excel(ausgabe_pfad, index=False)
88     '''
89     Dies speichert den sortierten DataFrame in eine neue Excel-Datei namens '
90     Fallzahlen_sortiert.xlsx'. Der Parameter 'index=False' sorgt dafür, dass
91     die DataFrame-Indizes nicht mitgespeichert werden.
92
93     ### Ausführen des Skripts:
94
95     Speichere das oben stehende Skript in einer Datei, z.B. 'fallzahlen_sortieren
96     .py', und führe es mit Python aus:
97
98     '''bash
99     python fallzahlen_sortieren.py
100     '''
101
102     ### Beispielausgabe:
103
104     '''
105     Excel-Datei erfolgreich eingelesen.
106
107     Erste 5 Zeilen der Originaldaten:
108
109     LOR-Schlüssel      Bezirke      Straftaten_insgesamt  Raub
110     Straßenraub  Körperverletzungen_insgesamt  ...  Branddelikte_insgesamt
111     Brandstiftung  Sachbeschädigung_insgesamt  Sachbeschädigung_Graffiti
112     Rauschgiftdelikte  Kieztaten
113
114     0      10000      Mitte      83611      884
115     521      8144      ...      420
116     137      6172      976
117     3306      19212
118
119     1      20000      Friedrichshain-Kreuzberg      55838      923
120     606      5502      ...      298
121     85      4906      801
122     4455      12724
123
124     ...      ...      ...      ...
125     ...      ...      ...      ...
126     ...      ...      ...      ...
127     ...      ...
128
129     [14 rows x 19 columns]
130
131     Anzahl der Zeilen nach dem Ausschluss: 12
132
133     Erste 5 Zeilen der sortierten Daten:
134
135     LOR-Schlüssel      Bezirke      Straftaten_insgesamt  Raub
136     Straßenraub  Körperverletzungen_insgesamt  ...  Branddelikte_insgesamt
137     Brandstiftung  Sachbeschädigung_insgesamt  Sachbeschädigung_Graffiti
138     Rauschgiftdelikte  Kieztaten
139
140     0      10000      Mitte      83611      884
141     521      8144      ...      420
142     137      6172      976

```

```

117 1      3306      19212      Friedrichshain-Kreuzberg      55838      923
      606      5502 ...      298
      85      4906      801
      4455      12724
118 ...      ...      ...      ...
      ...      ...
      ...      ...
      ...      ...
119
120 [12 rows x 19 columns]
121
122 Sortierte Daten wurden erfolgreich in 'Fallzahlen_sortiert.xlsx' gespeichert.
123 '''
124
125 Dieses Skript liest die Excel-Datei ein, filtert die unerwünschten Zeilen
      heraus, sortiert die verbleibenden Daten nach der Gesamtzahl der
      Straftaten und speichert das Ergebnis in einer neuen Excel-Datei. Du
      kannst das Skript nach Bedarf anpassen, z.B. die Sortierreihenfolge ändern
      oder weitere Analysen hinzufügen.

```

Listing 75: AI Response Testfall 1 Prompt 2 Ausführung 5

```

1 Natürlich! Hier ist ein robustes Python-Skript, das die beschriebenen
      Schritte ausführt. Das Skript verwendet die 'pandas'-Bibliothek, um die
      Excel-Datei zu verarbeiten und sicherzustellen, dass potenzielle Fehler
      wie fehlende Dateien, fehlende Blätter oder fehlende Spalten behandelt
      werden.
2
3 ### Voraussetzungen
4
5 Stellen Sie sicher, dass die 'pandas'-Bibliothek installiert ist. Falls nicht
      , können Sie sie mit folgendem Befehl installieren:
6
7 '''bash
8 pip install pandas
9 '''
10
11 Zusätzlich benötigt 'pandas' zum Lesen von Excel-Dateien die 'openpyxl'-
      Bibliothek. Installieren Sie diese gegebenenfalls ebenfalls:
12
13 '''bash
14 pip install openpyxl
15 '''
16
17 ### Python-Skript
18
19 '''python
20 import pandas as pd
21 import sys
22
23 def read_excel_file(file_path, sheet_name):
24     """

```



```

25     Liest eine Excel-Datei und gibt das entsprechende Sheet als DataFrame zur
26     ück.
27     """
28     try:
29         df = pd.read_excel(file_path, sheet_name=sheet_name)
30         print(f"Erfolgreich das Blatt '{sheet_name}' aus '{file_path}'
31         gelesen.")
32         return df
33     except FileNotFoundError:
34         print(f"Fehler: Die Datei '{file_path}' wurde nicht gefunden.")
35         sys.exit(1)
36     except ValueError:
37         print(f"Fehler: Das Blatt '{sheet_name}' existiert nicht in '{
38         file_path}'")
39         sys.exit(1)
40     except Exception as e:
41         print(f"Ein unerwarteter Fehler ist aufgetreten: {e}")
42         sys.exit(1)
43
44 def validate_columns(df, required_columns):
45     """
46     Überprüft, ob die erforderlichen Spalten im DataFrame vorhanden sind.
47     """
48     missing_columns = [col for col in required_columns if col not in df.
49     columns]
50     if missing_columns:
51         print(f"Fehler: Die folgenden erforderlichen Spalten fehlen im
52         DataFrame: {missing_columns}")
53         sys.exit(1)
54     else:
55         print("Alle erforderlichen Spalten sind vorhanden.")
56
57 def sort_dataframe(df, sort_column, exclude_keys, key_column):
58     """
59     Sortiert den DataFrame nach 'sort_column' absteigend, wobei die Zeilen
60     mit 'exclude_keys'
61     am Ende verbleiben.
62     """
63     # Trennen der Zeilen, die ausgeschlossen werden sollen
64     df_excluded = df[df[key_column].isin(exclude_keys)]
65     df_to_sort = df[~df[key_column].isin(exclude_keys)]
66
67     # Sortieren der relevanten Zeilen
68     df_sorted = df_to_sort.sort_values(by=sort_column, ascending=False)
69
70     # Zusammenführen der sortierten und ausgeschlossenen Zeilen
71     df_final = pd.concat([df_sorted, df_excluded], ignore_index=True)
72
73     print("DataFrame erfolgreich sortiert.")
74     return df_final
75
76 def main():

```

```

71 # Parameter
72 file_path = 'Fallzahlen.xlsx'
73 sheet_name = 'Fallzahlen_2023'
74 sort_column = 'Straftaten_insgesamt'
75 key_column = 'LOR-Schlüssel'
76 exclude_keys = [999900, 999999]
77 output_file = 'Fallzahlen_sortiert.xlsx'
78
79 # Schritt 1: Daten einlesen
80 df = read_excel_file(file_path, sheet_name)
81
82 # Schritt 2: Validierung der erforderlichen Spalten
83 required_columns = [sort_column, key_column]
84 validate_columns(df, required_columns)
85
86 # Schritt 3: Sortierung durchführen
87 df_sorted = sort_dataframe(df, sort_column, exclude_keys, key_column)
88
89 # Optional: Weitere Überprüfung, ob alle Zeilen enthalten sind
90 if len(df_sorted) != len(df):
91     print("Warnung: Die Anzahl der Zeilen nach der Sortierung stimmt
nicht mit der ursprünglichen Anzahl überein.")
92
93 # Schritt 4: Ergebnis speichern
94 try:
95     df_sorted.to_excel(output_file, index=False)
96     print(f"Das sortierte DataFrame wurde erfolgreich in '{output_file}'
gespeichert.")
97 except Exception as e:
98     print(f"Fehler beim Speichern der Datei: {e}")
99     sys.exit(1)
100
101 if __name__ == "__main__":
102     main()
103 '''
104
105 ### Erklärung des Skripts
106
107 1. **Importieren der benötigten Bibliotheken:**
108     - 'pandas' für die Datenverarbeitung.
109     - 'sys' für das Beenden des Skripts bei Fehlern.
110
111 2. **Funktion 'read_excel_file':**
112     - Versucht, die angegebene Excel-Datei und das Sheet zu lesen.
113     - Behandelt Fehler wie fehlende Dateien oder Blätter und gibt
entsprechende Fehlermeldungen aus.
114
115 3. **Funktion 'validate_columns':**
116     - Überprüft, ob die notwendigen Spalten ('Straftaten_insgesamt' und '
LOR-Schlüssel') im DataFrame vorhanden sind.
117     - Beendet das Skript mit einer Fehlermeldung, wenn eine erforderliche
Spalte fehlt.

```

```

118
119 4. **Funktion 'sort_dataframe':**
120     - Trennt die Zeilen, die nicht sortiert werden sollen ('LOR-Schlüssel'
121       999900 und 999999).
122     - Sortiert die verbleibenden Zeilen nach 'Straftaten_insgesamt'
123       absteigend.
124     - Fügt die ausgeschlossenen Zeilen am Ende des DataFrames hinzu.
125
126 5. **Funktion 'main':**
127     - Definiert die Parameter wie Dateipfad, Sheet-Name, Spaltennamen und
128       Ausgabedateinamen.
129     - Führt die Schritte des Einlesens, Validierens, Sortierens und Speicherns
130       durch.
131     - Gibt Erfolgsmeldungen oder Fehlermeldungen aus.
132
133 6. **Ausführung des Skripts:**
134     - Das Skript startet mit der Ausführung der 'main'-Funktion.
135
136 ### Nutzung des Skripts
137
138 1. **Speichern Sie das Skript:**
139     - Speichern Sie den obigen Code in einer Python-Datei, z.B. '
140       sort_fallzahlen.py'.
141
142 2. **Platzieren Sie die Excel-Datei:**
143     - Stellen Sie sicher, dass sich 'Fallzahlen.xlsx' im gleichen Verzeichnis
144       wie das Skript befindet oder passen Sie den 'file_path' entsprechend an.
145
146 3. **Führen Sie das Skript aus:**
147     - Öffnen Sie ein Terminal oder eine Eingabeaufforderung.
148     - Navigieren Sie zum Verzeichnis des Skripts.
149     - Führen Sie das Skript mit folgendem Befehl aus:
150
151       '''
152       bash
153       python sort_fallzahlen.py
154       '''
155
156 4. **Ergebnis:**
157     - Das sortierte DataFrame wird in einer neuen Excel-Datei namens '
158       Fallzahlen_sortiert.xlsx' gespeichert.
159     - Bei Fehlern erhalten Sie entsprechende Fehlermeldungen im Terminal.
160
161 ### Anpassungen
162
163 - **Andere Sheet-Namen oder Dateipfade:**
164     - Passen Sie die Variablen 'file_path' und 'sheet_name' im 'main'-Bereich
165       entsprechend an.
166
167 - **Weitere Sortierkriterien:**
168     - Wenn Sie nach zusätzlichen Kriterien sortieren möchten, können Sie die '
169       sort_values'-Funktion entsprechend erweitern.
170

```

```

161 - **Weitere Fehlersicherungen:**
162   - Sie können zusätzliche Überprüfungen hinzufügen, z.B. überprüfen, ob die
      auszuschließenden Schlüssel tatsächlich vorhanden sind.
163
164 Ich hoffe, dieses Skript hilft Ihnen weiter! Falls Sie weitere Anpassungen
      oder Erklärungen benötigen, lassen Sie es mich wissen.

```

Listing 76: AI Response Testfall 1 Prompt 3 Ausführung 1

```

1 Natürlich, hier ist ein robustes Python-Skript, das die von Ihnen
  beschriebenen Schritte durchführt. Das Skript liest die Excel-Datei '
  Fallzahlen.xlsx', sortiert die Daten nach der Spalte 'Straftaten_insgesamt'
  absteigend, wobei die Zeilen mit den LOR-Schlüsseln '999900' und
  '999999' am Ende bleiben, und speichert das Ergebnis in einem neuen Pandas
  DataFrame. Zudem enthält das Skript Fehlerbehandlungen für fehlende
  Spalten und andere potenzielle Probleme.
2
3 ### Benötigte Bibliotheken
4
5 Stellen Sie sicher, dass Sie die benötigten Bibliotheken installiert haben.
  Sie können 'pandas' und 'openpyxl' (für das Lesen von Excel-Dateien) mit
  pip installieren:
6
7 ```bash
8 pip install pandas openpyxl
9 ```
10
11 ### Python-Skript
12
13 ```python
14 import pandas as pd
15
16 def sort_fallzahlen_excel(file_path: str, sheet_name: str) -> pd.DataFrame:
17     """
18     Liest die Excel-Datei, sortiert die Daten nach 'Straftaten_insgesamt'
19     absteigend,
20     wobei die Zeilen mit den LOR-Schlüsseln 999900 und 999999 am Ende bleiben
21     .
22
23     Parameters:
24     - file_path: Pfad zur Excel-Datei.
25     - sheet_name: Name des Sheets in der Excel-Datei.
26
27     Returns:
28     - Sortierter Pandas DataFrame.
29     """
30     try:
31         # Schritt 1: Einlesen der Excel-Datei
32         df = pd.read_excel(file_path, sheet_name=sheet_name)
33         print(f"Excel-Datei '{file_path}' erfolgreich eingelesen.")
34
35         # Überprüfen, ob die notwendigen Spalten vorhanden sind
36         benötigte_spalten = ['LOR-Schlüssel', 'Straftaten_insgesamt']

```

```

35     fehlende_spalten = [spalte for spalte in benötigte_spalten if spalte
36     not in df.columns]
37     if fehlende_spalten:
38         raise ValueError(f"Fehlende Spalten in der Excel-Datei: {
39     fehlende_spalten}")
40
41     # Schritt 2: Trennen der Bezirke und Nicht-Bezirke
42     nicht_bezirke_keys = [999900, 999999]
43
44     # Sicherstellen, dass 'LOR-Schlüssel' als numerisch betrachtet wird
45     df['LOR-Schlüssel'] = pd.to_numeric(df['LOR-Schlüssel'], errors='
46     coerce')
47
48     bezirke_df = df[~df['LOR-Schlüssel'].isin(nicht_bezirke_keys)].copy()
49     nicht_bezirke_df = df[df['LOR-Schlüssel'].isin(nicht_bezirke_keys)].
50     copy()
51
52     # Schritt 3: Sortieren der Bezirke nach 'Straftaten_insgesamt'
53     absteigend
54     bezirke_df.sort_values(by='Straftaten_insgesamt', ascending=False,
55     inplace=True)
56
57     # Schritt 4: Zusammenfügen der sortierten Bezirke mit den Nicht-
58     Bezirken
59     sortiertes_df = pd.concat([bezirke_df, nicht_bezirke_df],
60     ignore_index=True)
61
62     print("Daten erfolgreich sortiert.")
63     return sortiertes_df
64
65     except FileNotFoundError:
66         print(f"Datei '{file_path}' wurde nicht gefunden.")
67     except ValueError as ve:
68         print(f"Wertfehler: {ve}")
69     except Exception as e:
70         print(f"Ein unerwarteter Fehler ist aufgetreten: {e}")
71
72 # Anwendung des Skripts
73 if __name__ == "__main__":
74     datei_pfad = 'Fallzahlen.xlsx'
75     sheet = 'Fallzahlen_2023'
76
77     sortiertes_dataframe = sort_fallzahlen_excel(datei_pfad, sheet)
78
79     if sortiertes_dataframe is not None:
80         # Schritt 5: Optional - Speichern des sortierten DataFrames in einer
81         neuen Excel-Datei
82         ausgabe_pfad = 'Sortierte_Fallzahlen.xlsx'
83         try:
84             sortiertes_dataframe.to_excel(ausgabe_pfad, index=False)
85             print(f"Sortiertes DataFrame wurde erfolgreich in '{ausgabe_pfad
86             }' gespeichert.")

```

```

77         except Exception as e:
78             print(f"Fehler beim Speichern der Datei: {e}")
79     '''
80
81     ### Erklärung der Schritte
82
83     1. **Einlesen der Excel-Datei:**
84         - Das Skript verwendet 'pd.read_excel', um das spezifizierte Sheet der
85           Excel-Datei zu lesen.
86         - Es werden Fehlermeldungen ausgegeben, falls die Datei nicht gefunden
87           wird oder das Sheet nicht existiert.
88
89     2. **Überprüfen der notwendigen Spalten:**
90         - Es wird sichergestellt, dass die Spalten 'LOR-Schlüssel' und '
91           Straftaten_insgesamt' vorhanden sind. Falls nicht, wird ein Fehler
92           ausgegeben.
93
94     3. **Trennen der Bezirke und Nicht-Bezirke:**
95         - Die Zeilen mit den LOR-Schlüsseln '999900' und '999999' werden als Nicht-
96           Bezirke identifiziert und separat gespeichert.
97
98     4. **Sortieren der Bezirke:**
99         - Die Bezirke werden nach der Spalte 'Straftaten_insgesamt' in
100           absteigender Reihenfolge sortiert.
101
102     5. **Zusammenfügen der DataFrames:**
103         - Der sortierte Bezirke-DataFrame wird mit dem Nicht-Bezirke-DataFrame
104           verbunden, sodass letztere am Ende stehen.
105
106     6. **Speichern des sortierten DataFrames:**
107         - Optional können Sie das sortierte DataFrame in einer neuen Excel-Datei
108           speichern.
109
110     ### Hinweise zur Robustheit
111
112     - **Fehlerbehandlung:** Das Skript enthält mehrere 'try-except'-Blöcke, um hä
113       ufig auftretende Fehler wie fehlende Dateien oder Spalten zu handhaben.
114     - **Datentypkonvertierung:** Es wird sichergestellt, dass die 'LOR-Schlüssel'
115       numerisch sind, um die Filterung korrekt durchführen zu können.
116     - **Flexibilität:** Das Skript ist so gestaltet, dass es leicht an andere
117       Dateien oder Sheets angepasst werden kann, indem die Variablen 'datei_pfad'
118       und 'sheet' geändert werden.
119
120     ### Ausführen des Skripts
121
122     Speichern Sie das Skript in einer '.py'-Datei, beispielsweise '
123       sort_fallzahlen.py', und führen Sie es mit Python aus:
124
125     '''bash
126     python sort_fallzahlen.py
127     '''

```

```

116 Stellen Sie sicher, dass sich die Excel-Datei 'Fallzahlen.xlsx' im gleichen
    Verzeichnis wie das Skript befindet oder geben Sie den vollständigen Pfad
    zur Datei an.
117
118 ---
119
120 Ich hoffe, dieses Skript erfüllt Ihre Anforderungen. Bei weiteren Fragen oder
    Anpassungswünschen stehe ich gerne zur Verfügung!

```

Listing 77: AI Response Testfall 1 Prompt 3 Ausführung 2

```

1  '''python
2  import pandas as pd
3  import sys
4
5  def main():
6      excel_file = 'Fallzahlen.xlsx'
7      sheet_name = 'Fallzahlen_2023'
8      lor_keys_to_exclude = [999900, 999999]
9
10     try:
11         # Schritt 1: Einlesen der Excel-Datei
12         df = pd.read_excel(excel_file, sheet_name=sheet_name)
13         print("Excel-Datei erfolgreich eingelesen.")
14     except FileNotFoundError:
15         print(f"Fehler: Die Datei '{excel_file}' wurde nicht gefunden.")
16         sys.exit(1)
17     except ValueError as e:
18         print(f"Fehler beim Einlesen des Sheets '{sheet_name}': {e}")
19         sys.exit(1)
20     except Exception as e:
21         print(f"Ein unerwarteter Fehler ist aufgetreten: {e}")
22         sys.exit(1)
23
24     # Überprüfen, ob die notwendigen Spalten vorhanden sind
25     required_columns = ['LOR-Schlüssel', 'Straftaten_insgesamt']
26     missing_columns = [col for col in required_columns if col not in df.
27                        columns]
28     if missing_columns:
29         print(f"Fehler: Fehlende Spalten in den Daten: {missing_columns}")
30         sys.exit(1)
31
32     # Sicherstellen, dass 'LOR-Schlüssel' numerisch ist
33     try:
34         df['LOR-Schlüssel'] = pd.to_numeric(df['LOR-Schlüssel'])
35     except ValueError:
36         print("Fehler: Die Spalte 'LOR-Schlüssel' enthält nicht-numerische
37            Werte.")
38         sys.exit(1)
39
40     # Schritt 2: Trennen der Bezirke und der auszuschließenden Schlüssel
41     df_bezirke = df[~df['LOR-Schlüssel'].isin(lor_keys_to_exclude)]
42     df_excluded = df[df['LOR-Schlüssel'].isin(lor_keys_to_exclude)]

```

```

41
42 # Schritt 3: Sortieren der Bezirke nach 'Straftaten_insgesamt' absteigend
43 if 'Straftaten_insgesamt' not in df_bezirke.columns:
44     print("Fehler: Die Spalte 'Straftaten_insgesamt' ist nicht in den
Daten vorhanden.")
45     sys.exit(1)
46
47 try:
48     df_bezirke_sorted = df_bezirke.sort_values(by='Straftaten_insgesamt',
ascending=False)
49 except Exception as e:
50     print(f"Fehler beim Sortieren der Daten: {e}")
51     sys.exit(1)
52
53 # Schritt 4: Zusammenführen der sortierten Bezirke mit den
ausgeschlossenen Zeilen
54 df_final = pd.concat([df_bezirke_sorted, df_excluded], ignore_index=True)
55
56 # Optional: Ausgabe oder Speicherung des finalen DataFrames
57 # Beispiel: Anzeigen der ersten 5 Zeilen
58 print("Sortiertes DataFrame:")
59 print(df_final.head())
60
61 # Beispiel: Speichern in eine neue Excel-Datei
62 output_file = 'Fallzahlen_sortiert.xlsx'
63 try:
64     df_final.to_excel(output_file, index=False)
65     print(f"Das sortierte DataFrame wurde erfolgreich in '{output_file}'
gespeichert.")
66 except Exception as e:
67     print(f"Fehler beim Speichern der Datei '{output_file}': {e}")
68     sys.exit(1)
69
70 if __name__ == "__main__":
71     main()
72 '''
73
74 ### Erklärung des Skripts
75
76 1. **Import der notwendigen Module:**
77 - 'pandas' für die Datenverarbeitung.
78 - 'sys' für das Beenden des Skripts bei Fehlern.
79
80 2. **Einlesen der Excel-Datei:**
81 - Versucht, die Datei 'Fallzahlen.xlsx' und das Sheet 'Fallzahlen_2023'
einzulesen.
82 - Fehlerbehandlung für häufige Fehler wie Datei nicht gefunden oder
falscher Sheet-Name.
83
84 3. **Überprüfen der notwendigen Spalten:**
85 - Sicherstellt, dass sowohl 'LOR-Schlüssel' als auch '
Straftaten_insgesamt' in den Daten vorhanden sind.

```



```

86 - Bricht das Skript ab, wenn Spalten fehlen.
87
88 4. **Datenvorbereitung:**
89 - Konvertiert die 'LOR-Schlüssel'-Spalte in numerische Werte, um die
90 Ausschlussbedingungen korrekt anwenden zu können.
91 - Trennt die DataFrame in zwei Teile:
92   - 'df_bezirke': alle Zeilen außer denen mit 'LOR-Schlüssel' 999900 und
93     999999.
94   - 'df_excluded': die auszuschließenden Zeilen.
95
96 5. **Sortieren der Bezirke:**
97 - Sortiert 'df_bezirke' nach 'Straftaten_insgesamt' in absteigender
98 Reihenfolge.
99 - Fehlerbehandlung für mögliche Probleme beim Sortieren.
100
101 6. **Zusammenführen der DataFrames:**
102 - Kombiniert das sortierte Bezirke-DataFrame mit den ausgeschlossenen
103 Zeilen am Ende.
104
105 7. **Optional: Ausgabe oder Speicherung:**
106 - Zeigt die ersten fünf Zeilen des finalen DataFrames an.
107 - Speichert das sortierte DataFrame in eine neue Excel-Datei 'Fallzahlen_sortiert.xlsx'.
108 - Fehlerbehandlung beim Speichern der Datei.
109
110 ### Hinweise
111
112 - **Robustheit:** Das Skript enthält umfassende Fehlerbehandlungen, die
113 sicherstellen, dass fehlende Dateien, fehlende Spalten oder Datenprobleme
114 ordnungsgemäß gehandhabt werden.
115
116 - **Anpassung:** Sie können den 'output_file' Namen ändern oder weitere
117 Verarbeitungsschritte hinzufügen, je nach Bedarf.
118
119 - **Ausführung:** Stellen Sie sicher, dass Sie die notwendigen Python-Pakete
120 installiert haben, insbesondere 'pandas' und 'openpyxl' (für das Lesen und
121 Schreiben von Excel-Dateien). Sie können fehlende Pakete mit 'pip install
122 pandas openpyxl' installieren.
123
124 ### Beispiel Ausführung
125
126 Nachdem Sie das Skript gespeichert haben (z.B. als 'sort_fallzahlen.py'), kö
127 nnen Sie es über die Kommandozeile ausführen:
128
129 '''bash
130 python sort_fallzahlen.py
131 '''

```

126 Wenn alles korrekt eingerichtet ist, wird das sortierte DataFrame erstellt  
und in der Datei 'Fallzahlen\_sortiert.xlsx' gespeichert.

Listing 78: AI Response Testfall 1 Prompt 3 Ausführung 3

```
1  '''python
2  import pandas as pd
3
4  def sort_fallzahlen_excel(file_path: str, sheet_name: str) -> pd.DataFrame:
5      """
6      Reads the specified sheet from an Excel file, sorts the data by '
7      Straftaten_insgesamt'
8      in descending order excluding rows with LOR-Schlüssel 999900 and 999999,
9      which are
10     appended at the end.
11
12     Parameters:
13     - file_path: Path to the Excel file.
14     - sheet_name: Name of the sheet to read.
15
16     Returns:
17     - A sorted pandas DataFrame.
18     """
19     try:
20         # Read the Excel sheet
21         df = pd.read_excel(file_path, sheet_name=sheet_name)
22         print(f"Successfully read the sheet '{sheet_name}' from '{file_path}'")
23     except FileNotFoundError:
24         print(f"Error: The file '{file_path}' was not found.")
25         return pd.DataFrame()
26     except ValueError:
27         print(f"Error: The sheet '{sheet_name}' does not exist in '{file_path}'")
28         return pd.DataFrame()
29     except Exception as e:
30         print(f"An unexpected error occurred while reading the Excel file: {e}")
31         return pd.DataFrame()
32
33     # Check if necessary columns exist
34     required_columns = ['LOR-Schlüssel', 'Straftaten_insgesamt']
35     missing_columns = [col for col in required_columns if col not in df.columns]
36     if missing_columns:
37         print(f"Error: The following required columns are missing from the data: {missing_columns}")
38         return pd.DataFrame()
39
40     # Separate the rows with LOR-Schlüssel 999900 and 999999
41     exclusion_keys = [999900, 999999]
42     excluded_df = df[df['LOR-Schlüssel'].isin(exclusion_keys)]
43     included_df = df[~df['LOR-Schlüssel'].isin(exclusion_keys)]
```

```

42
43     if excluded_df.empty:
44         print("Warning: No rows found with LOR-Schlüssel 999900 or 999999.")
45
46     # Sort the included DataFrame by 'Straftaten_insgesamt' in descending
47     order
48     try:
49         included_sorted_df = included_df.sort_values(by='Straftaten_insgesamt', ascending=False)
50         print("Successfully sorted the included data by 'Straftaten_insgesamt' in descending order.")
51     except KeyError:
52         print("Error: The column 'Straftaten_insgesamt' does not exist in the data.")
53         return pd.DataFrame()
54     except Exception as e:
55         print(f"An unexpected error occurred during sorting: {e}")
56         return pd.DataFrame()
57
58     # Concatenate the sorted included data with the excluded data
59     sorted_df = pd.concat([included_sorted_df, excluded_df], ignore_index=True)
60     print("Successfully concatenated the excluded rows at the end of the sorted data.")
61
62     return sorted_df
63
64 if __name__ == "__main__":
65     # Define the Excel file path and sheet name
66     excel_file = 'Fallzahlen.xlsx'
67     sheet = 'Fallzahlen_2023'
68
69     # Get the sorted DataFrame
70     sorted_fallzahlen_df = sort_fallzahlen_excel(excel_file, sheet)
71
72     # Check if the DataFrame is not empty before proceeding
73     if not sorted_fallzahlen_df.empty:
74         print("Sorted DataFrame:")
75         print(sorted_fallzahlen_df)
76         # Optionally, you can save the sorted DataFrame to a new Excel file or another format
77         # sorted_fallzahlen_df.to_excel('Sorted_Fallzahlen.xlsx', index=False)
78     else:
79         print("No data to display due to previous errors.")
80
81     """
82
83     ### Erklärung des Skripts
84
85     1. **Importieren der notwendigen Bibliotheken:**
86     - 'pandas' wird benötigt, um die Excel-Datei zu lesen und mit DataFrames zu arbeiten.

```

```

85
86 2. **Funktion 'sort_fallzahlen_excel':**
87   - **Parameter:**
88     - 'file_path': Der Pfad zur Excel-Datei ('Fallzahlen.xlsx').
89     - 'sheet_name': Der Name des Sheets ('Fallzahlen_2023').
90   - **Schritte innerhalb der Funktion:**
91     - **Excel-Datei einlesen:**
92       - Versucht, das angegebene Sheet aus der Excel-Datei zu lesen.
93       - Behandelt mögliche Fehler wie fehlende Datei oder fehlendes Sheet.
94     - **Überprüfen der erforderlichen Spalten:**
95       - Stellt sicher, dass sowohl 'LOR-Schlüssel' als auch '
Straftaten_insgesamt' in den Daten vorhanden sind.
96       - Gibt eine Fehlermeldung aus, wenn eine dieser Spalten fehlt.
97     - **Ausschließen der spezifischen Zeilen:**
98       - Trennt die Zeilen mit den 'LOR-Schlüssel'-Werten '999900' und
'999999' vom Rest des DataFrames.
99       - Diese Zeilen sollen am Ende stehen bleiben.
100    - **Sortieren der verbleibenden Daten:**
101      - Sortiert die verbleibenden Zeilen nach 'Straftaten_insgesamt' in
absteigender Reihenfolge.
102    - **Kombinieren der sortierten Daten:**
103      - Fügt die sortierten Daten und die ausgeschlossenen Zeilen zu einem
neuen DataFrame zusammen.
104
105 3. **Main-Block ('if __name__ == "__main__":'):**
106    - Definiert den Dateinamen und den Sheet-Namen.
107    - Ruft die Funktion 'sort_fallzahlen_excel' auf, um das sortierte
DataFrame zu erhalten.
108    - Überprüft, ob das zurückgegebene DataFrame nicht leer ist, bevor es
ausgegeben wird.
109    - Optional kann das sortierte DataFrame in eine neue Excel-Datei
gespeichert werden (kommentierte Zeile).
110
111 ### Hinweise zur Robustheit
112
113 - **Fehlerbehandlung:**
114   - Das Skript behandelt verschiedene potenzielle Fehler, einschließlich
fehlender Dateien, fehlender Sheets und fehlender Spalten.
115   - Informative Fehlermeldungen werden ausgegeben, um dem Benutzer zu helfen,
das Problem zu verstehen.
116
117 - **Flexibilität:**
118   - Die Funktion 'sort_fallzahlen_excel' ist so gestaltet, dass sie für
verschiedene Dateien und Sheets wiederverwendet werden kann, indem einfach
die Parameter geändert werden.
119
120 - **Erweiterbarkeit:**
121   - Weitere Validierungen oder Verarbeitungsschritte können leicht hinzugefü
gt werden, falls erforderlich.
122
123 ### Ausführung des Skripts
124

```

- 125 1. Stellen Sie sicher, dass die Datei `'Fallzahlen.xlsx'` im gleichen Verzeichnis wie das Skript liegt oder geben Sie den vollständigen Pfad zur Datei an.
- 126
- 127 2. Führen Sie das Skript mit Python aus:
- 128 `'''bash`
- 129 `python sort_fallzahlen.py`
- 130 `'''`
- 131 (Angenommen, Sie speichern das Skript als `'sort_fallzahlen.py'`)
- 132
- 133 3. Das Skript gibt das sortierte DataFrame `in` der Konsole aus. Optional können Sie das Ergebnis `in` eine neue Datei speichern, indem Sie die entsprechende Zeile entkommentieren.

Listing 79: AI Response Testfall 1 Prompt 3 Ausführung 4

```

1 Hier ist ein robustes Python-Skript, das die von dir beschriebenen Schritte
  ausführt. Es verwendet die 'pandas'-Bibliothek zum Einlesen und
  Verarbeiten der Excel-Datei. Das Skript beinhaltet umfassende
  Fehlerbehandlungen, um sicherzustellen, dass fehlende Spalten oder andere
  potenzielle Probleme angemessen behandelt werden.
2
3 '''python
4 import pandas as pd
5
6 def sort_fallzahlen_excel(file_path: str, sheet_name: str) -> pd.DataFrame:
7     """
8     Liest die Daten aus dem angegebenen Excel-Sheet, sortiert sie nach '
9     Straftaten_insgesamt' absteigend,
10    wobei die Zeilen mit den LOR-Schlüsseln 999900 und 999999 am Ende des
11    DataFrames stehen.
12
13    Parameters:
14    - file_path: Pfad zur Excel-Datei.
15    - sheet_name: Name des Sheets, das eingelesen werden soll.
16
17    Returns:
18    - Ein sortierter Pandas DataFrame.
19    """
20    try:
21        # Schritt 1: Einlesen der Excel-Datei
22        df = pd.read_excel(file_path, sheet_name=sheet_name)
23        print(f"Erfolgreich das Sheet '{sheet_name}' aus '{file_path}'
24        eingelesen.")
25    except FileNotFoundError:
26        raise FileNotFoundError(f"Die Datei '{file_path}' wurde nicht
27        gefunden.")
28    except ValueError:
29        raise ValueError(f"Das Sheet '{sheet_name}' existiert nicht in '{
30        file_path}'")
31    except Exception as e:
32        raise Exception(f"Ein unerwarteter Fehler ist beim Einlesen der Excel
33        -Datei aufgetreten: {e}")

```

```

28
29 # Überprüfen, ob die notwendigen Spalten vorhanden sind
30 required_columns = ['LOR-Schlüssel', 'Straftaten_insgesamt']
31 missing_columns = [col for col in required_columns if col not in df.
32 columns]
33 if missing_columns:
34     raise KeyError(f"Die folgenden erforderlichen Spalten fehlen im
35 DataFrame: {missing_columns}")
36 else:
37     print("Alle erforderlichen Spalten sind vorhanden.")
38
39 # Schritt 2: Separiere die Bezirke und die nicht zuzuordnenden Einträge
40 # Definiere die LOR-Schlüssel, die nicht sortiert werden sollen
41 exclude_lor_keys = [999900, 999999]
42
43 # Konvertiere 'LOR-Schlüssel' zu integer, falls nötig
44 if df['LOR-Schlüssel'].dtype != 'int64' and df['LOR-Schlüssel'].dtype !=
45 'int32':
46     try:
47         df['LOR-Schlüssel'] = df['LOR-Schlüssel'].astype(int)
48         print("'LOR-Schlüssel' erfolgreich in Integer konvertiert.")
49     except ValueError:
50         raise ValueError("'LOR-Schlüssel' enthält nicht-numerische Werte,
51 die nicht in Integer konvertiert werden können.")
52
53 # Teile den DataFrame in Bezirke und Nicht-Bezirke
54 bezirke_df = df[~df['LOR-Schlüssel'].isin(exclude_lor_keys)]
55 nicht_bezirke_df = df[df['LOR-Schlüssel'].isin(exclude_lor_keys)]
56
57 print(f"Anzahl der Bezirke: {bezirke_df.shape[0]}")
58 print(f"Anzahl der nicht zuzuordnenden Einträge: {nicht_bezirke_df.shape
59 [0]}")
60
61 # Schritt 3: Sortiere die Bezirke nach 'Straftaten_insgesamt' absteigend
62 try:
63     bezirke_sorted = bezirke_df.sort_values(by='Straftaten_insgesamt',
64 ascending=False)
65     print("Bezirke erfolgreich nach 'Straftaten_insgesamt' absteigend
66 sortiert.")
67 except KeyError:
68     raise KeyError("Die Spalte 'Straftaten_insgesamt' wurde nicht
69 gefunden und kann nicht zum Sortieren verwendet werden.")
70 except Exception as e:
71     raise Exception(f"Ein unerwarteter Fehler ist beim Sortieren der
72 Bezirke aufgetreten: {e}")
73
74 # Schritt 4: Konkateniere die sortierten Bezirke mit den nicht
75 zuzuordnenden Einträgen
76 final_df = pd.concat([bezirke_sorted, nicht_bezirke_df], ignore_index=
77 True)
78 print("Bezirke und nicht zuzuordnende Einträge erfolgreich kombiniert.")
79

```

```

69     return final_df
70
71 if __name__ == "__main__":
72     # Definiere den Pfad zur Excel-Datei und den Sheet-Namen
73     excel_file = 'Fallzahlen.xlsx'
74     sheet = 'Fallzahlen_2023'
75
76     try:
77         # Führe die Sortierung durch und hole das sortierte DataFrame
78         sortiertes_df = sort_fallzahlen_excel(excel_file, sheet)
79         print("Daten erfolgreich sortiert.")
80
81         # Optional: Anzeige der ersten paar Zeilen des sortierten DataFrames
82         print(sortiertes_df.head())
83
84         # Optional: Speichern des sortierten DataFrames in eine neue Excel-
85         # Datei
86         # sortiertes_df.to_excel('Fallzahlen_sortiert.xlsx', index=False)
87         # print("Sortiertes DataFrame wurde in 'Fallzahlen_sortiert.xlsx'
88         # gespeichert.")
89
90     except Exception as e:
91         print(f"Fehler: {e}")
92
93
94 ### Erklärung des Skripts
95
96 1. **Importieren der notwendigen Bibliothek**:
97     '''python
98     import pandas as pd
99     '''
100
101 2. **Definition der Funktion 'sort_fallzahlen_excel'**:
102     - **Parameter**:
103         - 'file_path': Der Pfad zur Excel-Datei ('Fallzahlen.xlsx').
104         - 'sheet_name': Der Name des Sheets, das eingelesen werden soll ('Fallzahlen_2023').
105     - **Schritte innerhalb der Funktion**:
106         - **Einlesen der Excel-Datei**: Verwendet 'pd.read_excel' und behandelt mögliche Fehler wie fehlende Dateien oder Sheets.
107         - **Überprüfung der erforderlichen Spalten**: Stellt sicher, dass sowohl 'LOR-Schlüssel' als auch 'Straftaten_insgesamt' vorhanden sind.
108         - **Konvertierung des 'LOR-Schlüssel'**: Stellt sicher, dass die LOR-Schlüssel als Ganzzahlen vorliegen, um die Filterung korrekt durchführen zu können.
109         - **Trennen der Bezirke und Nicht-Bezirke**: Teilt den DataFrame in zwei Teile - Bezirke und die speziellen Einträge mit den LOR-Schlüsseln '999900' und '999999'.
110         - **Sortieren der Bezirke**: Sortiert die Bezirke basierend auf 'Straftaten_insgesamt' in absteigender Reihenfolge.
111         - **Kombinieren der DataFrames**: Fügt die sortierten Bezirke und die Nicht-Bezirke wieder zusammen.

```

```

110
111 3. **Ausführung des Skripts**:
112     - **Definieren des Excel-Dateipfads und Blattnamens**:
113         python
114         excel_file = 'Fallzahlen.xlsx'
115         sheet = 'Fallzahlen_2023'
116         '''
117     - **Aufrufen der Funktion und Handhaben von Fehlern**:
118         - Versucht, das sortierte DataFrame zu erhalten und gibt den Erfolg
119         oder auftretende Fehler aus.
120     - **Optionale Schritte**:
121         - Anzeigen der ersten paar Zeilen des sortierten DataFrames mit '
122         print(sortiertes_df.head())'.
123         - Speichern des sortierten DataFrames in eine neue Excel-Datei durch
124         Entkommentieren der entsprechenden Zeilen.
125
126 ### Hinweise zur Robustheit
127
128     - **Fehlerbehandlung**: Das Skript fängt verschiedene potenzielle Fehler ab,
129     wie fehlende Dateien, Sheets oder Spalten und gibt klare Fehlermeldungen
130     aus.
131     - **Datentyp-Prüfung**: Stellt sicher, dass die 'LOR-Schlüssel'-Spalte
132     numerisch ist, bevor sie für die Filterung verwendet wird.
133     - **Flexibilität**: Die Funktion 'sort_fallzahlen_excel' kann leicht für
134     andere Dateien oder Sheets angepasst werden, indem einfach die Parameter
135     geändert werden.
136
137 ### Voraussetzungen
138
139     Stelle sicher, dass die 'pandas'-Bibliothek installiert ist. Falls nicht,
140     kannst du sie mit folgendem Befehl installieren:
141
142     '''bash
143     pip install pandas
144     '''
145
146     Zusätzlich benötigt 'pandas' die 'openpyxl'-Bibliothek zum Lesen von Excel-
147     Dateien. Diese wird normalerweise automatisch mit 'pandas' installiert,
148     andernfalls kannst du sie separat installieren:
149
150     '''bash
151     pip install openpyxl
152     '''
153
154 ### Ausführung des Skripts
155
156     Speichere das Skript in einer Python-Datei, beispielsweise 'sort_fallzahlen.
157     py', und führe es über die Kommandozeile aus:
158
159     '''bash
160     python sort_fallzahlen.py
161     '''

```



```

150
151 Das Skript liest die Daten, sortiert sie wie gewünscht und gibt den Erfolg
    sowie eine Vorschau der sortierten Daten aus. Optional kannst du das
    sortierte DataFrame in eine neue Excel-Datei speichern, indem du die
    entsprechenden Zeilen im Skript entkommentierst.

```

Listing 80: AI Response Testfall 1 Prompt 3 Ausführung 5

```

1 Hier ist ein Python-Skript, das alle Sheets aus der Excel-Datei 'Fallzahlen.
  xlsx' einliest, die Daten kombiniert, nach der Anzahl der insgesamt
  erfassten Straftaten ('Straftaten_insgesamt') sortiert und in einem Pandas
  DataFrame speichert.
2
3 ### Voraussetzungen
4
5 Stelle sicher, dass du die benötigten Bibliotheken installiert hast. Du
  kannst sie mit 'pip' installieren:
6
7 '''bash
8 pip install pandas openpyxl
9 '''
10
11 ### Python-Skript
12
13 '''python
14 import pandas as pd
15
16 def lese_und_sortiere_fallzahlen(excel_datei):
17     """
18     Liest alle Sheets aus der angegebenen Excel-Datei, kombiniert die Daten,
19     sortiert sie nach 'Straftaten_insgesamt' in absteigender Reihenfolge
20     und gibt den sortierten DataFrame zurück.
21
22     :param excel_datei: Pfad zur Excel-Datei (z.B. 'Fallzahlen.xlsx')
23     :return: Sortierter Pandas DataFrame
24     """
25     try:
26         # Alle Sheets der Excel-Datei lesen. sheet_name=None liest alle
        Sheets als Dict.
27         alle_sheets = pd.read_excel(excel_datei, sheet_name=None)
28
29         # Liste zur Speicherung aller DataFrames
30         dataframe_liste = []
31
32         # Durch alle Sheets iterieren und die DataFrames zur Liste hinzufügen
33         for sheet_name, df in alle_sheets.items():
34             # Optional: Hinzufügen einer Spalte mit dem Sheet-Namen, falls nö
            tig
35             # df['Sheet_Name'] = sheet_name
36             dataframe_liste.append(df)
37
38         # Alle DataFrames zu einem einzigen DataFrame kombinieren
39         kombiniertes_df = pd.concat(dataframe_liste, ignore_index=True)

```

```

40
41     # Prüfen, ob die Spalte 'Straftaten_insgesamt' existiert
42     if 'Straftaten_insgesamt' not in kombiniertes_df.columns:
43         raise ValueError("Die Spalte 'Straftaten_insgesamt' wurde in den
44         Daten nicht gefunden.")
45
46     # Sortieren nach 'Straftaten_insgesamt' in absteigender Reihenfolge
47     sortiertes_df = kombiniertes_df.sort_values(by='Straftaten_insgesamt',
48     , ascending=False).reset_index(drop=True)
49
50     return sortiertes_df
51
52 except FileNotFoundError:
53     print(f"Die Datei {excel_datei} wurde nicht gefunden.")
54 except Exception as e:
55     print(f"Ein Fehler ist aufgetreten: {e}")
56
57 if __name__ == "__main__":
58     # Pfad zur Excel-Datei
59     dateipfad = 'Fallzahlen.xlsx'
60
61     # Funktion aufrufen und sortierten DataFrame erhalten
62     df_sortiert = lese_und_sortiere_fallzahlen(dateipfad)
63
64     if df_sortiert is not None:
65         # Sortierten DataFrame anzeigen
66         print(df_sortiert)
67
68         # Optional: Sortierten DataFrame in eine neue Excel-Datei speichern
69         # df_sortiert.to_excel('Fallzahlen_sortiert.xlsx', index=False)
70
71     '''
72
73     ### Erklärung des Skripts
74
75     1. **Importieren der Bibliotheken**: Das Skript verwendet 'pandas' zum
76     Verarbeiten der Excel-Daten.
77
78     2. **Funktion 'lese_und_sortiere_fallzahlen'**:
79     - **Lesen aller Sheets**: Mit 'pd.read_excel(excel_datei, sheet_name=None
80     )' werden alle Sheets der Excel-Datei als ein Dictionary eingelesen, wobei
81     die Schlüssel die Sheet-Namen sind und die Werte die entsprechenden
82     DataFrames.
83     - **Kombinieren der DataFrames**: Alle einzelnen DataFrames werden in
84     eine Liste ('dataframe_liste') gesammelt und anschließend mit 'pd.concat'
85     zu einem großen DataFrame ('kombiniertes_df') kombiniert.
86     - **Sortieren**: Der kombinierte DataFrame wird nach der Spalte '
87     Straftaten_insgesamt' in absteigender Reihenfolge sortiert.
88     - **Fehlerbehandlung**: Es gibt grundlegende Fehlerprüfungen, um
89     sicherzustellen, dass die Datei existiert und die erforderliche Spalte
90     vorhanden ist.
91
92     3. **Ausführen der Funktion**: Im '__main__'-Block wird die Funktion

```

```

aufgerufen, und der sortierte DataFrame wird angezeigt. Optional kann der
sortierte DataFrame auch in eine neue Excel-Datei gespeichert werden (der
entsprechende Code ist auskommentiert).
81
82 ### Weitere Anpassungen
83
84 - **Sheet-Namen beibehalten**: Falls du den Ursprung der Daten aus den
    einzelnen Sheets nachvollziehen möchtest, könntest du eine zusätzliche
    Spalte mit dem Sheet-Namen hinzufügen (siehe auskommentierte Zeile im
    Skript).
85
86 - **Datenbereinigung**: Je nach Datenqualität kannst du weitere Bereinigungen
    oder Transformationen vornehmen, bevor du die Daten kombinierst oder
    sortierst.
87
88 ### Ausführen des Skripts
89
90 Speichere das Skript in einer Datei, z.B. 'sortiere_fallzahlen.py', und führe
    es in deinem Terminal oder deiner IDE aus:
91
92 ```bash
93 python sortiere_fallzahlen.py
94 ```
95
96 Das Skript wird die kombinierten und sortierten Daten in der Konsole anzeigen
    . Wenn du die sortierten Daten in eine neue Excel-Datei speichern möchtest
    , entferne den Kommentar vor der entsprechenden Zeile im Skript.

```

Listing 81: AI Response Testfall 2 Prompt 1 Ausführung 1

```

1 Hier ist ein Python-Skript, das die Anforderungen erfüllt. Es liest alle
    Sheets der Excel-Datei 'Fallzahlen.xlsx', aggregiert die Daten nach
    Bezirken basierend auf der Anzahl der 'Straftaten_insgesamt', sortiert die
    Bezirke absteigend nach der Gesamtanzahl der Straftaten und speichert das
    Ergebnis in einem Pandas DataFrame.
2
3 Stelle sicher, dass du die erforderlichen Bibliotheken installiert hast.
    Falls nicht, kannst du sie mit 'pip' installieren:
4
5 ```bash
6 pip install pandas openpyxl
7 ```
8
9 Hier ist das vollständige Skript:
10
11 ```python
12 import pandas as pd
13
14 # Pfad zur Excel-Datei
15 excel_datei = 'Fallzahlen.xlsx'
16
17 # Einlesen aller Sheets in ein Dictionary von DataFrames
18 try:

```

```

19     alle_sheets = pd.read_excel(excel_datei, sheet_name=None)
20 except FileNotFoundError:
21     print(f"Die Datei {excel_datei} wurde nicht gefunden.")
22     exit(1)
23 except Exception as e:
24     print(f"Fehler beim Einlesen der Excel-Datei: {e}")
25     exit(1)
26
27 # Liste zur Speicherung der relevanten Daten aus allen Sheets
28 daten_liste = []
29
30 # Durch alle DataFrames iterieren und relevante Daten extrahieren
31 for sheet_name, df in alle_sheets.items():
32     # Überprüfen, ob die notwendigen Spalten vorhanden sind
33     if 'Bezirke' in df.columns and 'Straftaten_insgesamt' in df.columns:
34         # Auswahl der relevanten Spalten
35         df_relevant = df[['Bezirke', 'Straftaten_insgesamt']]
36         daten_liste.append(df_relevant)
37     else:
38         print(f"Sheet '{sheet_name}' enthält nicht die erforderlichen Spalten")
39
40 # Kombinieren aller relevanten Daten
41 if daten_liste:
42     alle_daten = pd.concat(daten_liste)
43 else:
44     print("Keine relevanten Daten gefunden.")
45     exit(1)
46
47 # Aggregation der Straftaten pro Bezirk
48 aggregierte_daten = alle_daten.groupby('Bezirke', as_index=False)['Straftaten_insgesamt'].sum()
49
50 # Sortieren der Bezirke nach der Anzahl der Straftaten (absteigend)
51 aggregierte_daten_sortiert = aggregierte_daten.sort_values(by='Straftaten_insgesamt', ascending=False).reset_index(drop=True)
52
53 # Optional: Anzeige des resultierenden DataFrames
54 print(aggregierte_daten_sortiert)
55
56 # Optional: Speichern des DataFrames in eine neue Excel-Datei
57 # aggregierte_daten_sortiert.to_excel('Aggregierte_Fallzahlen.xlsx', index=False)
58 '''
59
60 ### Erklärung des Skripts:
61
62 1. **Importieren der Bibliotheken:**
63     - 'pandas' wird zum Verarbeiten der Excel-Datei verwendet.
64
65 2. **Einlesen der Excel-Datei:**
66     - Mit 'pd.read_excel' und 'sheet_name=None' werden alle Sheets als

```

```

Dictionary von DataFrames eingelesen.
67 - Fehlerbehandlung stellt sicher, dass das Skript bei Problemen mit dem
Einlesen der Datei nicht abstürzt.
68
69 3. **Extrahieren relevanter Daten:**
70 - Für jedes Sheet wird geprüft, ob die benötigten Spalten ('Bezirke' und '
Straftaten_insgesamt') vorhanden sind.
71 - Die relevanten Spalten werden extrahiert und zur Liste 'daten_liste'
hinzugefügt.
72 - Sheets, die die erforderlichen Spalten nicht enthalten, werden ü
bersprungen und eine Warnung wird ausgegeben.
73
74 4. **Kombinieren der Daten:**
75 - Alle relevanten DataFrames werden mittels 'pd.concat' zu einem einzigen
DataFrame 'alle_daten' kombiniert.
76
77 5. **Aggregieren der Straftaten pro Bezirk:**
78 - Mit 'groupby' und 'sum' werden die Straftaten pro Bezirk summiert.
79
80 6. **Sortieren der Bezirke:**
81 - Der aggregierte DataFrame wird nach 'Straftaten_insgesamt' in
absteigender Reihenfolge sortiert.
82
83 7. **Ausgabe und Speichern:**
84 - Der resultierende DataFrame wird ausgegeben.
85 - Optional kann der DataFrame in eine neue Excel-Datei gespeichert werden,
indem die entsprechende Zeile entkommentiert wird.
86
87 ### Hinweise:
88
89 - **Anpassungen bei den Spaltennamen:**
90 Stelle sicher, dass die Spaltennamen exakt mit denen in deiner Excel-Datei
übereinstimmen, einschließlich Groß-/Kleinschreibung und eventueller
Leerzeichen.
91
92 - **Speichern der Ergebnisse:**
93 Wenn du das Ergebnis in einer neuen Excel-Datei speichern möchtest,
entferne den Kommentar ('#') vor der letzten Zeile.
94
95 - **Leistung bei großen Dateien:**
96 Bei sehr großen Excel-Dateien kann das Skript mehr Speicher und Zeit benö
tigen. In solchen Fällen könnte eine optimierte Datenverarbeitung
notwendig sein.
97
98 Ich hoffe, dieses Skript hilft dir weiter! Falls du weitere Anpassungen benö
tigtst oder Fragen hast, lass es mich wissen.

```

Listing 82: AI Response Testfall 2 Prompt 1 Ausführung 2

```

1 Hier ist ein Python-Skript, das alle Sheets der Excel-Datei 'Fallzahlen.xlsx
' einliest, die Daten nach der Gesamtanzahl der Straftaten pro Bezirk
aggregiert, sortiert und in einem Pandas DataFrame speichert:
2

```

```

3  '''python
4  import pandas as pd
5
6  # Pfad zur Excel-Datei
7  excel_datei = 'Fallzahlen.xlsx'
8
9  # Alle Sheet-Namen aus der Excel-Datei auslesen
10 xls = pd.ExcelFile(excel_datei)
11 sheet_namen = xls.sheet_names
12
13 # Liste zur Speicherung der DataFrames aller Sheets
14 daten_frames = []
15
16 # Iteriere über jedes Sheet und lese die Daten ein
17 for sheet in sheet_namen:
18     df = pd.read_excel(excel_datei, sheet_name=sheet)
19     daten_frames.append(df)
20
21 # Alle DataFrames zu einem einzigen DataFrame zusammenführen
22 kombiniert_df = pd.concat(daten_frames, ignore_index=True)
23
24 # Optional: Duplikate entfernen, falls Bezirke in mehreren Sheets vorkommen
25 # kombiniert_df = kombiniert_df.drop_duplicates(subset=['Bezirke'])
26
27 # Gruppieren nach 'Bezirke' und Summieren der 'Straftaten_insgesamt'
28 aggregiert_df = kombiniert_df.groupby('Bezirke', as_index=False)['
    Straftaten_insgesamt'].sum()
29
30 # Sortieren nach der Anzahl der Straftaten in absteigender Reihenfolge
31 sortiert_df = aggregiert_df.sort_values(by='Straftaten_insgesamt', ascending=
    False)
32
33 # Optional: Index zurücksetzen
34 sortiert_df = sortiert_df.reset_index(drop=True)
35
36 # Ergebnis anzeigen
37 print(sortiert_df)
38
39 # Optional: Das sortierte Ergebnis in eine neue Excel- oder CSV-Datei
    speichern
40 # sortiert_df.to_excel('Aggregierte_Fallzahlen.xlsx', index=False)
41 # sortiert_df.to_csv('Aggregierte_Fallzahlen.csv', index=False)
42 '''
43
44 ### Erklärung der Schritte:
45
46 1. **Importieren der benötigten Bibliothek:**
47     '''python
48     import pandas as pd
49     '''
50     Wir verwenden 'pandas', eine leistungsstarke Bibliothek zur
        Datenmanipulation und -analyse.

```

```

51
52 2. **Laden der Excel-Datei und Ermitteln der Sheet-Namen:**
53     '''python
54     excel_datei = 'Fallzahlen.xlsx'
55     xls = pd.ExcelFile(excel_datei)
56     sheet_namen = xls.sheet_names
57     '''
58     Hier wird die Excel-Datei geöffnet und die Namen aller enthaltenen Sheets
        ausgelesen.
59
60 3. **Einlesen der Daten aus allen Sheets:**
61     '''python
62     daten_frames = []
63     for sheet in sheet_namen:
64         df = pd.read_excel(excel_datei, sheet_name=sheet)
65         daten_frames.append(df)
66     '''
67     Jedes Sheet wird einzeln als DataFrame eingelesen und in der Liste '
        daten_frames' gespeichert.
68
69 4. **Kombinieren aller DataFrames:**
70     '''python
71     kombiniert_df = pd.concat(daten_frames, ignore_index=True)
72     '''
73     Alle DataFrames werden zu einem einzigen DataFrame zusammengeführt. '
        ignore_index=True' sorgt dafür, dass die Indizes neu erstellt werden.
74
75 5. **Aggregieren der Straftaten pro Bezirk:**
76     '''python
77     aggregiert_df = kombiniert_df.groupby('Bezirke', as_index=False)['
        Straftaten_insgesamt'].sum()
78     '''
79     Die Daten werden nach dem Bezirk gruppiert und die Gesamtanzahl der
        Straftaten pro Bezirk summiert.
80
81 6. **Sortieren der Daten:**
82     '''python
83     sortiert_df = aggregiert_df.sort_values(by='Straftaten_insgesamt',
        ascending=False)
84     sortiert_df = sortiert_df.reset_index(drop=True)
85     '''
86     Der DataFrame wird in absteigender Reihenfolge nach der Anzahl der
        Straftaten sortiert. 'reset_index' wird verwendet, um die Indizes nach dem
        Sortieren neu zu setzen.
87
88 7. **Anzeigen des Ergebnisses:**
89     '''python
90     print(sortiert_df)
91     '''
92     Das sortierte DataFrame wird in der Konsole angezeigt.
93
94 8. **Optional: Speichern des Ergebnisses:**

```

```

95     '''python
96     # sortiert_df.to_excel('Aggregierte_Fallzahlen.xlsx', index=False)
97     # sortiert_df.to_csv('Aggregierte_Fallzahlen.csv', index=False)
98     '''
99     Falls gewünscht, kann das Ergebnis in eine neue Excel- oder CSV-Datei
    gespeichert werden. Die entsprechenden Zeilen können entkommentiert werden
    , um diese Funktionalität zu aktivieren.
100
101     ### Hinweise:
102
103     - **Duplikate:** Falls ein Bezirk in mehreren Sheets vorkommt und du die
    Einträge nicht aggregieren möchtest, kannst du die Zeile zur Entfernung
    von Duplikaten ('drop_duplicates') nutzen. Andernfalls aggregierst du die
    Straftaten pro Bezirk unabhängig von den einzelnen Sheets.
104
105     - **Weitere Analysen:** Du kannst den 'aggregiert_df' oder 'sortiert_df'
    weitere Analysen unterziehen oder zusätzliche Spalten berechnen, je nach
    Bedarf.
106
107     - **Fehlerbehandlung:** Für größere Projekte oder zur Produktion könnten zusätz-
    liche Schritte zur Fehlerbehandlung sinnvoll sein, z.B. Überprüfen, ob
    bestimmte Spalten existieren oder mit fehlenden Daten umgehen.
108
109     Dieses Skript sollte dir einen guten Startpunkt bieten, um die Daten aus
    deiner Excel-Datei zu verarbeiten und die gewünschten Informationen zu
    extrahieren.

```

Listing 83: AI Response Testfall 2 Prompt 1 Ausführung 3

```

1  Gerne! Hier ist ein Python-Skript, das alle Sheets aus der Excel-Datei '
    Fallzahlen.xlsx' liest, die Daten nach der Anzahl der Straftaten pro
    Bezirk aggregiert, sortiert und in einem Pandas DataFrame speichert.
2
3  '''python
4  import pandas as pd
5
6  # Pfad zur Excel-Datei
7  excel_file = 'Fallzahlen.xlsx'
8
9  # Schritt 1: Alle Sheets der Excel-Datei einlesen
10 # Das Argument sheet_name=None liest alle Sheets und gibt ein Dictionary zurück
11 all_sheets = pd.read_excel(excel_file, sheet_name=None)
12
13 # Schritt 2: Daten aus allen Sheets zusammenführen
14 # Wir gehen davon aus, dass alle Sheets die gleiche Struktur haben
15 # Daher können wir sie einfach mit pd.concat zusammenfügen
16 combined_df = pd.concat(all_sheets.values(), ignore_index=True)
17
18 # Schritt 3: Daten nach 'Bezirke' gruppieren und 'Straftaten_insgesamt'
    summieren
19 # Dies aggregiert die Straftaten pro Bezirk über alle Sheets hinweg
20 grouped_df = combined_df.groupby('Bezirke')['Straftaten_insgesamt'].sum().

```



```

    reset_index()
21
22 # Schritt 4: Die aggregierten Daten nach 'Straftaten_insgesamt' sortieren (
    absteigend)
23 sorted_df = grouped_df.sort_values(by='Straftaten_insgesamt', ascending=False
    ).reset_index(drop=True)
24
25 # Optional: Anzeigen des sortierten DataFrames
26 print(sorted_df)
27
28 # Optional: Den sortierten DataFrame in eine neue Excel-Datei speichern
29 sorted_df.to_excel('Aggregierte_Straftaten_pro_Bezirk.xlsx', index=False)
30 '''
31
32 ### Erklärung der einzelnen Schritte:
33
34 1. **Import der notwendigen Bibliothek:**
35     '''python
36     import pandas as pd
37     '''
38     Wir verwenden die 'pandas'-Bibliothek, die leistungsstarke Datenstrukturen
        und Datenanalysetools bietet.
39
40 2. **Einlesen aller Sheets der Excel-Datei:**
41     '''python
42     all_sheets = pd.read_excel(excel_file, sheet_name=None)
43     '''
44     - 'sheet_name=None' sorgt dafür, dass alle Sheets als ein Dictionary
        eingelesen werden, wobei die Schlüssel die Sheet-Namen und die Werte die
        entsprechenden DataFrames sind.
45
46 3. **Zusammenführen der Daten aus allen Sheets:**
47     '''python
48     combined_df = pd.concat(all_sheets.values(), ignore_index=True)
49     '''
50     - 'pd.concat' verbindet alle DataFrames vertikal (also untereinander).
51     - 'ignore_index=True' stellt sicher, dass die Indizes im kombinierten
        DataFrame neu nummeriert werden.
52
53 4. **Gruppieren und Aggregieren der Straftaten pro Bezirk:**
54     '''python
55     grouped_df = combined_df.groupby('Bezirke')['Straftaten_insgesamt'].sum().
        reset_index()
56     '''
57     - 'groupby('Bezirke')' gruppiert die Daten nach dem Bezirk.
58     - '['Straftaten_insgesamt'].sum()' summiert die Straftaten innerhalb jeder
        Gruppe.
59     - 'reset_index()' wandelt das Ergebnis zurück in ein reguläres DataFrame.
60
61 5. **Sortieren der aggregierten Daten:**
62     '''python
63     sorted_df = grouped_df.sort_values(by='Straftaten_insgesamt', ascending=

```

```

False).reset_index(drop=True)
64 '''
65 - 'sort_values' sortiert die DataFrame-Zeilen basierend auf der
    angegebenen Spalte.
66 - 'ascending=False' sortiert die Daten in absteigender Reihenfolge.
67 - 'reset_index(drop=True)' entfernt den alten Index und ersetzt ihn durch
    einen neuen fortlaufenden Index.
68
69 6. **Optional: Anzeige und Speicherung des Ergebnisses:**
70 '''python
71 print(sorted_df)
72 sorted_df.to_excel('Aggregierte_Straftaten_pro_Bezirk.xlsx', index=False)
73 '''
74 - 'print(sorted_df)' zeigt den finalen DataFrame an.
75 - 'to_excel' speichert den DataFrame in einer neuen Excel-Datei ohne den
    Index.
76
77 ### Hinweise:
78
79 - **Abhängigkeiten installieren:** Stelle sicher, dass 'pandas' installiert
    ist. Falls nicht, kannst du es mit 'pip' installieren:
80 '''bash
81 pip install pandas
82 '''
83
84 - **Excel-Engine:** Standardmäßig verwendet 'pandas' die 'openpyxl'-Engine fü
    r '.xlsx'-Dateien. Falls du eine ältere Excel-Version verwendest oder
    Probleme auftreten, kannst du die Engine explizit angeben:
85 '''python
86 all_sheets = pd.read_excel(excel_file, sheet_name=None, engine='openpyxl')
87 '''
88
89 - **Fehlerbehandlung:** Für eine robuste Anwendung könntest du zusätzliche
    Fehlerprüfungen einbauen, z.B. ob die Datei existiert, ob die erwarteten
    Spalten vorhanden sind usw.
90
91 Dieses Skript sollte deinen Anforderungen entsprechen und dir eine
    aggregierte Übersicht der Straftaten pro Bezirk aus allen Sheets deiner
    Excel-Datei liefern.

```

Listing 84: AI Response Testfall 2 Prompt 1 Ausführung 4

```

1 Natürlich! Hier ist ein vollständiges Python-Skript, das die beschriebenen
    Anforderungen erfüllt. Das Skript verwendet die 'pandas'-Bibliothek, um
    alle Sheets der Excel-Datei 'Fallzahlen.xlsx' zu lesen, die Daten zu
    kombinieren, nach der Gesamtanzahl der Straftaten pro Bezirk zu sortieren
    und das Ergebnis in einem 'DataFrame' zu speichern.
2
3 ### Schritt-für-Schritt-Anleitung:
4
5 1. **Installiere die benötigten Bibliotheken:** Stelle sicher, dass 'pandas'
    und 'openpyxl' installiert sind. 'openpyxl' wird von 'pandas' verwendet,
    um Excel-Dateien zu lesen.

```

```

6
7     '''bash
8     pip install pandas openpyxl
9     '''
10
11 2. **Erstelle das Python-Skript**:
12
13     '''python
14     import pandas as pd
15
16     def lese_und_sortiere_fallzahlen(excel_datei):
17         """
18         Liest alle Sheets aus der gegebenen Excel-Datei, kombiniert die Daten
19         ,
20         sortiert sie nach der Gesamtanzahl der Straftaten pro Bezirk und
21         gibt das sortierte DataFrame zurück.
22
23         :param excel_datei: Pfad zur Excel-Datei 'Fallzahlen.xlsx'
24         :return: Sortiertes pandas DataFrame
25         """
26         try:
27             # Alle Sheets lesen; sheet_name=None gibt ein Dictionary zurück
28             alle_sheets = pd.read_excel(excel_datei, sheet_name=None)
29             print(f"Anzahl der gelesenen Sheets: {len(alle_sheets)}")
30         except FileNotFoundError:
31             print(f"Die Datei '{excel_datei}' wurde nicht gefunden.")
32             return None
33         except Exception as e:
34             print(f"Ein Fehler ist aufgetreten: {e}")
35             return None
36
37         # Kombinieren aller DataFrames in einen
38         try:
39             kombiniertes_df = pd.concat(alle_sheets.values(), ignore_index=
40 True)
41             print(f"Anzahl der Zeilen nach dem Kombinieren: {len(
42 kombiniertes_df)}")
43         except ValueError as ve:
44             print(f"Fehler beim Kombinieren der Daten: {ve}")
45             return None
46
47         # Sicherstellen, dass die Spalte 'Straftaten_insgesamt' numerisch ist
48         if not pd.api.types.is_numeric_dtype(kombiniertes_df['
49 Straftaten_insgesamt']):
50             kombiniertes_df['Straftaten_insgesamt'] = pd.to_numeric(
51                 kombiniertes_df['Straftaten_insgesamt'], errors='coerce'
52             )
53             kombiniertes_df = kombiniertes_df.dropna(subset=['
54 Straftaten_insgesamt'])
55
56         # Sortieren nach 'Straftaten_insgesamt' absteigend
57         sortiertes_df = kombiniertes_df.sort_values(

```

```

53         by='Straftaten_insgesamt', ascending=False
54     ).reset_index(drop=True)
55
56     print("Die Daten wurden erfolgreich sortiert.")
57     return sortiertes_df
58
59 def main():
60     excel_datei = 'Fallzahlen.xlsx'
61     df = lese_und_sortiere_fallzahlen(excel_datei)
62     if df is not None:
63         # Optional: Das sortierte DataFrame anzeigen
64         print(df)
65
66         # Optional: Das sortierte DataFrame speichern
67         # df.to_csv('Sortierte_Fallzahlen.csv', index=False, encoding='
utf-8-sig')
68         # print("Das sortierte DataFrame wurde als 'Sortierte_Fallzahlen.
csv' gespeichert.")
69
70     if __name__ == "__main__":
71         main()
72     '''
73
74 ### Erklärung des Skripts:
75
76 1. **Importieren der Bibliothek**:
77     - 'pandas' wird für die Datenmanipulation und das Lesen der Excel-Datei
verwendet.
78
79 2. **Funktion 'lese_und_sortiere_fallzahlen'**:
80     - **Lesen aller Sheets**: Mit 'pd.read_excel()' und 'sheet_name=None'
werden alle Sheets der Excel-Datei in ein Dictionary geladen.
81     - **Kombinieren der DataFrames**: Mit 'pd.concat()' werden alle
DataFrames aus den einzelnen Sheets zu einem einzigen DataFrame kombiniert.
82     - **Datenbereinigung**: Es wird sichergestellt, dass die Spalte '
Straftaten_insgesamt' numerisch ist. Falls nicht, werden nicht
konvertierbare Werte entfernt.
83     - **Sortieren**: Das kombinierte DataFrame wird nach der Spalte '
Straftaten_insgesamt' in absteigender Reihenfolge sortiert.
84
85 3. **Funktion 'main'**:
86     - Ruft die Funktion zum Lesen und Sortieren auf.
87     - Gibt das sortierte DataFrame aus.
88     - Optional kann das sortierte DataFrame auch in eine CSV-Datei
gespeichert werden (auskommentiert im Skript).
89
90 4. **Ausführung des Skripts**:
91     - Wenn das Skript direkt ausgeführt wird, startet die 'main()'-Funktion.
92
93 ### Nutzung:
94

```

```

95 1. **Speichere das Skript** beispielsweise als 'sortiere_fallzahlen.py'.
96 2. **Stelle sicher**, dass sich die Datei 'Fallzahlen.xlsx' im gleichen
    Verzeichnis wie das Skript befindet oder passe den Pfad entsprechend an.
97 3. **Führe das Skript aus**:
98
99     '''bash
100     python sortiere_fallzahlen.py
101     '''
102
103 4. **Optional**: Wenn du das sortierte Ergebnis auch als CSV-Datei speichern
    möchtest, entferne das Kommentarzeichen ('#') vor den entsprechenden
    Zeilen im Skript.
104
105 ### Beispielausgabe:
106
107 Die Ausgabe wird ein 'DataFrame' sein, das nach der Gesamtzahl der Straftaten
    pro Bezirk sortiert ist, beginnend mit dem Bezirk mit den meisten
    Straftaten. Hier ist eine vereinfachte Darstellung basierend auf deinem
    Beispiel:
108
109 '''
110
111      LOR-Schlüssel          Bezirke  Straftaten_insgesamt  Raub
112      Straßenraub ...  Rauschgiftdelikte  Kieztaten
113 0      999999          Berlin (PKS gesamt)          536695  5389
114      3333 ...          17502          119737
115 1      10000          Mitte          83611  884
116      521 ...          3306          19212
117 2      80000          Neukölln          45313  508
118      294 ...          1869          10253
119 ...
120 '''
121
122 Dieses Skript sollte dir eine solide Grundlage bieten, um deine Daten
    effizient zu verarbeiten und nach den gewünschten Kriterien zu sortieren.
    Wenn du weitere Anpassungen benötigst, lass es mich wissen!

```

Listing 85: AI Response Testfall 2 Prompt 1 Ausführung 5

```

1 Natürlich, ich helfe dir gerne dabei, ein Python-Skript zu erstellen, das
    deine Anforderungen erfüllt. Das Skript wird die Daten aus allen Sheets
    der Excel-Datei 'Fallzahlen.xlsx' zusammenführen, die unerwünschten 'LOR-
    Schlüssel'-Zeilen entfernen, die Daten nach der Spalte '
    Straftaten_insgesamt' sortieren und das Ergebnis in einem Pandas DataFrame
    speichern.
2
3 Hier ist ein vollständiges Python-Skript, das diese Schritte ausführt:
4
5 '''python
6 import pandas as pd
7
8 def zusammenfuegen_und_sortieren(excel_datei):
9     """
10     Liest alle Sheets einer Excel-Datei, fügt sie zusammen,

```

```

11     entfernt bestimmte LOR-Schlüssel, sortiert nach 'Straftaten_insgesamt'
12     und gibt den resultierenden DataFrame zurück.
13
14     Parameters:
15     - excel_datei: str, Pfad zur Excel-Datei.
16
17     Returns:
18     - pd.DataFrame: Gefilterter und sortierter DataFrame.
19     """
20
21     # Laden Sie die Excel-Datei und holen Sie alle Sheet-Namen
22     xls = pd.ExcelFile(excel_datei)
23     sheet_namen = xls.sheet_names
24
25     # Liste zur Speicherung der DataFrames aus jedem Sheet
26     df_list = []
27
28     # Durchlaufen Sie jede Sheet und lesen Sie die Daten
29     for sheet in sheet_namen:
30         df = pd.read_excel(excel_datei, sheet_name=sheet)
31         df_list.append(df)
32         print(f"Gesheet '{sheet}' erfolgreich geladen.")
33
34     # Verbinden Sie alle DataFrames zu einem einzigen DataFrame
35     kombiniert_df = pd.concat(df_list, ignore_index=True)
36     print("Alle Sheets wurden erfolgreich zusammengeführt.")
37
38     # Entfernen Sie die Zeilen mit den LOR-Schlüsseln 999900 und 999999
39     filtered_df = kombiniert_df[~kombiniert_df['LOR-Schlüssel'].isin([999900,
40     999999])]
41     print("Unerwünschte LOR-Schlüssel wurden entfernt.")
42
43     # Sortieren Sie den DataFrame nach 'Straftaten_insgesamt' in absteigender
44     Reihenfolge
45     sorted_df = filtered_df.sort_values(by='Straftaten_insgesamt', ascending=
46     False).reset_index(drop=True)
47     print("DataFrame wurde nach 'Straftaten_insgesamt' sortiert.")
48
49     return sorted_df
50
51 if __name__ == "__main__":
52     # Pfad zur Excel-Datei
53     excel_pfad = 'Fallzahlen.xlsx'
54
55     # Rufen Sie die Funktion auf und speichern Sie das Ergebnis in 'final_df'
56     final_df = zusammenfuegen_und_sortieren(excel_pfad)
57
58     # Optional: Zeigen Sie die ersten paar Zeilen des finalen DataFrames an
59     print(final_df.head())
60
61     # Optional: Speichern Sie den finalen DataFrame als neue Excel-Datei
62     # final_df.to_excel('Zusammengefuegte_Fallzahlen.xlsx', index=False)

```

```

60 '''
61
62 ### Erklärung der Schritte:
63
64 1. **Importieren der notwendigen Bibliothek:**
65     '''python
66     import pandas as pd
67     '''
68
69 2. **Definieren einer Funktion 'zusammenfuegen_und_sortieren':**
70     - Diese Funktion übernimmt den Pfad zur Excel-Datei als Parameter.
71     - Sie liest alle Sheets der Excel-Datei und speichert sie in einer Liste
72       von DataFrames.
73     - Anschließend werden alle DataFrames zu einem einzigen DataFrame
74       kombiniert.
75
76 3. **Entfernen der unerwünschten 'LOR-Schlüssel':**
77     '''python
78     filtered_df = kombiniert_df[~kombiniert_df['LOR-Schlüssel'].isin([999900,
79     999999])]
80     '''
81     - Diese Zeile filtert alle Zeilen heraus, deren 'LOR-Schlüssel' entweder
82       '999900' oder '999999' ist.
83
84 4. **Sortieren des DataFrames:**
85     '''python
86     sorted_df = filtered_df.sort_values(by='Straftaten_insgesamt', ascending=
87     False).reset_index(drop=True)
88     '''
89     - Sortiert den DataFrame basierend auf der Spalte 'Straftaten_insgesamt'
90       in absteigender Reihenfolge.
91     - 'reset_index(drop=True)' setzt den Index zurück, sodass er fortlaufend
92       von 0 beginnt.
93
94 5. **Hauptblock des Skripts:**
95     - Definiert den Pfad zur Excel-Datei.
96     - Ruft die oben definierte Funktion auf und speichert das Ergebnis in '
97       final_df'.
98     - Optional zeigt es die ersten paar Zeilen des finalen DataFrames an.
99     - Optional kann der finale DataFrame in eine neue Excel-Datei gespeichert
100       werden (die entsprechende Zeile ist auskommentiert).
101
102 ### Hinweise:
103
104 - **Abhängigkeiten installieren:**
105     Stelle sicher, dass du die benötigten Bibliotheken installiert hast. Du
106     kannst 'pandas' mit folgendem Befehl installieren:
107     '''bash
108     pip install pandas
109     '''
110     Falls deine Excel-Datei im '.xlsx'-Format ist, wird 'openpyxl' automatisch
111     von 'pandas' verwendet. Falls nicht, installiere es mit:

```

```

101  '''bash
102  pip install openpyxl
103  '''
104
105  - **Anpassen des Pfades:**
106  Stelle sicher, dass der Pfad zur Excel-Datei ('Fallzahlen.xlsx') korrekt
    ist. Wenn die Datei sich in einem anderen Verzeichnis befindet, gib den
    vollständigen Pfad an.
107
108  - **Speichern des Ergebnisses:**
109  Falls du das Ergebnis in eine neue Excel-Datei speichern möchtest, entferne
    den Kommentar ('#') vor der entsprechenden Zeile:
110  '''python
111  final_df.to_excel('Zusammengefuegte_Fallzahlen.xlsx', index=False)
112  '''
113
114  Dieses Skript sollte deinen Anforderungen entsprechen und die gewünschten
    Daten verarbeiten. Bei weiteren Fragen oder Anpassungswünschen stehe ich
    gerne zur Verfügung!

```

Listing 86: AI Response Testfall 2 Prompt 2 Ausführung 1

```

1  Natürlich! Hier ist ein Python-Skript, das die von Ihnen beschriebenen
    Anforderungen erfüllt. Das Skript verwendet die 'pandas'-Bibliothek, um
    die Excel-Datei zu lesen, die Daten aus allen Sheets zusammenzuführen,
    unerwünschte Zeilen zu entfernen, die Daten nach der Spalte "
    Straftaten_insgesamt" zu sortieren und das Ergebnis in einem DataFrame zu
    speichern.
2
3  ### Voraussetzungen
4
5  Stellen Sie sicher, dass Sie die notwendigen Bibliotheken installiert haben.
    Sie können dies mit 'pip' tun:
6
7  '''bash
8  pip install pandas openpyxl
9  '''
10
11  ### Python-Skript
12
13  '''python
14  import pandas as pd
15
16  def load_and_process_excel(file_path):
17      """
18      Lädt alle Sheets aus der angegebenen Excel-Datei, fügt sie zusammen,
19      filtert unerwünschte Zeilen, sortiert die Daten und gibt einen DataFrame
      zurück.
20
21      Parameters:
22          file_path (str): Der Pfad zur Excel-Datei.
23
24      Returns:

```



```

25     pd.DataFrame: Der verarbeitete DataFrame.
26     """
27     try:
28         # Lade alle Sheets in ein Dictionary von DataFrames
29         all_sheets = pd.read_excel(file_path, sheet_name=None, dtype={'LOR-
        Schlüssel': str})
30
31         # Liste zum Speichern der DataFrames
32         df_list = []
33
34         for sheet_name, df in all_sheets.items():
35             print(f"Lade Sheet: {sheet_name} mit {len(df)} Zeilen.")
36             df_list.append(df)
37
38         # Kombiniere alle DataFrames in einen einzigen DataFrame
39         combined_df = pd.concat(df_list, ignore_index=True)
40         print(f"Gesamtanzahl der Zeilen nach dem Zusammenführen: {len(
        combined_df)}")
41
42         # Entferne Zeilen mit LOR-Schlüssel 999900 und 999999
43         filtered_df = combined_df[~combined_df['LOR-Schlüssel'].isin([999900,
        999999])]
44         print(f"Anzahl der Zeilen nach dem Filtern: {len(filtered_df)}")
45
46         # Sortiere nach 'Straftaten_insgesamt' absteigend
47         sorted_df = filtered_df.sort_values(by='Straftaten_insgesamt',
        ascending=False)
48         print("Daten wurden nach 'Straftaten_insgesamt' sortiert.")
49
50         # Optional: Setze den Index zurück
51         sorted_df.reset_index(drop=True, inplace=True)
52
53         return sorted_df
54
55     except FileNotFoundError:
56         print(f"Die Datei {file_path} wurde nicht gefunden.")
57     except Exception as e:
58         print(f"Ein Fehler ist aufgetreten: {e}")
59
60 if __name__ == "__main__":
61     # Pfad zur Excel-Datei
62     excel_file = 'Fallzahlen.xlsx'
63
64     # Verarbeite die Excel-Datei
65     final_df = load_and_process_excel(excel_file)
66
67     if final_df is not None:
68         # Zeige die ersten paar Zeilen des finalen DataFrames
69         print("\nErgebnis:")
70         print(final_df.head())
71
72         # Optional: Speichere den DataFrame in eine neue Excel- oder CSV-

```

```

73     Datei
74         # final_df.to_excel('Zusammengefuegte_Fallzahlen.xlsx', index=False)
75         # final_df.to_csv('Zusammengefuegte_Fallzahlen.csv', index=False)
76     '''
77     ### Erklärung der Schritte
78
79     1. **Importieren der Bibliotheken**:
80     - 'pandas' wird verwendet, um Excel-Dateien zu lesen und Daten zu
      verarbeiten.
81
82     2. **Funktion 'load_and_process_excel'**:
83     - **Laden aller Sheets**: Mit 'pd.read_excel' und 'sheet_name=None' werden
      alle Sheets der Excel-Datei in ein Dictionary geladen, wobei die Sheet-
      Namen die Schlüssel sind.
84     - **Zusammenführen der DataFrames**: Alle einzelnen DataFrames werden mit
      'pd.concat' zu einem großen DataFrame zusammengeführt.
85     - **Filtern der unerwünschten Zeilen**: Mit 'isin' werden die Zeilen mit
      den 'LOR-Schlüssel'-Werten '999900' und '999999' entfernt.
86     - **Sortieren der Daten**: Der DataFrame wird nach der Spalte "
      Straftaten_insgesamt" in absteigender Reihenfolge sortiert.
87     - **Zurücksetzen des Index**: Der Index wird zurückgesetzt, um konsistente
      Indizes im finalen DataFrame zu gewährleisten.
88
89     3. **Ausführen des Skripts**:
90     - Das Skript wird direkt ausgeführt, wenn es als Hauptprogramm gestartet
      wird.
91     - Nach der Verarbeitung werden die ersten paar Zeilen des finalen
      DataFrames angezeigt.
92     - Optional können Sie den finalen DataFrame in eine neue Excel- oder CSV-
      Datei speichern, indem Sie die entsprechenden Zeilen entkommentieren.
93
94     ### Zusätzliche Hinweise
95
96     - **Datentypen**: Ich habe 'dtype={'LOR-Schlüssel': str}' beim Laden der
      Excel-Datei hinzugefügt, um sicherzustellen, dass der 'LOR-Schlüssel' als
      String behandelt wird. Falls dieser als numerisch behandelt werden soll, k
      önnen Sie dies entsprechend anpassen.
97
98     - **Fehlerbehandlung**: Das Skript enthält grundlegende Fehlerbehandlungen fü
      r den Fall, dass die Datei nicht gefunden wird oder andere Fehler
      auftreten.
99
100    - **Performance**: Bei sehr großen Excel-Dateien kann das Zusammenführen und
      Verarbeiten eine Weile dauern. Stellen Sie sicher, dass Ihr System genü
      gend Ressourcen hat.
101
102    ### Beispielausgabe
103
104    Nach dem Ausführen des Skripts sollte eine ähnliche Ausgabe wie folgt
      angezeigt werden:
105

```

```

106 '''
107 Lade Sheet: Sheet1 mit 14 Zeilen.
108 Lade Sheet: Sheet2 mit 14 Zeilen.
109 ...
110 Gesamtanzahl der Zeilen nach dem Zusammenführen: 140
111 Anzahl der Zeilen nach dem Filtern: 138
112 Daten wurden nach 'Straftaten_insgesamt' sortiert.
113
114 Ergebnis:
115      LOR-Schlüssel                Bezirke  Straftaten_insgesamt  ...
116      Rauschgiftdelikte  Kieztaten
117 0      10000                Mitte                83611  ...
118      3306                19212
119 1      20000      Friedrichshain-Kreuzberg                55838
120      ...      4455                12724
121 2      80000                Neukölln                45313
122      ...      1869                10253
123 3      70000      Tempelhof-Schöneberg                41776
124      ...      1435                10132
125 4      100000      Marzahn-Hellersdorf                26649
126      ...      424                8203
127 '''
128
129 *Bitte beachten Sie, dass die tatsächlichen Werte je nach Ihren Daten
130 variieren.*
131
132 ### Anpassungen nach Bedarf
133
134 - **Speichern des DataFrames**: Falls Sie den finalen DataFrame speichern mö
135   chten, können Sie die entsprechenden Zeilen im Skript entkommentieren.
136
137 - **Weitere Filterungen oder Transformationen**: Je nach Ihren Anforderungen
138   können Sie weitere Filterbedingungen oder Datenmanipulationen hinzufügen.
139
140 Ich hoffe, dieses Skript hilft Ihnen weiter! Wenn Sie weitere Fragen haben
141 oder Anpassungen benötigen, lassen Sie es mich gerne wissen.

```

Listing 87: AI Response Testfall 2 Prompt 2 Ausführung 2

```

1 Natürlich! Hier ist ein Python-Skript, das deine Anforderungen erfüllt. Es
2 verwendet die 'pandas' Bibliothek, um die Excel-Datei zu verarbeiten, alle
3 relevanten Sheets zu laden, die unerwünschten Bezirke auszuschließen, die
4 Daten zusammenzuführen und nach der Spalte 'Straftaten_insgesamt' zu
5 sortieren.
6
7 ### Voraussetzungen:
8
9 Stelle sicher, dass du die 'pandas' und 'openpyxl' Bibliotheken installiert
10 hast. Du kannst sie mit folgendem Befehl installieren:
11
12 '''bash
13 pip install pandas openpyxl
14 '''

```

```

10
11 ### Python-Skript:
12
13 '''python
14 import pandas as pd
15
16 # Pfad zur Excel-Datei
17 excel_file = 'Fallzahlen.xlsx'
18
19 # LOR-Schlüssel, die ausgeschlossen werden sollen
20 exclude_lor_keys = [999900, 999999]
21
22 # Alle Sheets laden
23 sheets_dict = pd.read_excel(excel_file, sheet_name=None)
24
25 # Liste zur Speicherung der bereinigten DataFrames
26 filtered_dfs = []
27
28 for sheet_name, df in sheets_dict.items():
29     # Sicherstellen, dass die benötigten Spalten vorhanden sind
30     if 'LOR-Schlüssel' not in df.columns or 'Bezirke' not in df.columns:
31         print(f"Sheet '{sheet_name}' übersprungen, da erforderliche Spalten
32             fehlen.")
33         continue
34
35     # Ausschließen der unerwünschten LOR-Schlüssel
36     df_filtered = df[~df['LOR-Schlüssel'].isin(exclude_lor_keys)].copy()
37
38     # Optional: Hinzufügen einer Spalte zur Kennzeichnung des Sheets
39     df_filtered['Sheet'] = sheet_name
40
41     filtered_dfs.append(df_filtered)
42
43 # Überprüfen, ob es DataFrames zum Zusammenführen gibt
44 if not filtered_dfs:
45     raise ValueError("Keine gültigen Daten zum Zusammenführen gefunden.")
46
47 # Zusammenführen aller DataFrames auf Basis von 'LOR-Schlüssel' und 'Bezirke'
48 # Falls es gemeinsame Spalten außer den Schlüsseln gibt, werden sie mit
49 # Suffixen versehen
50 from functools import reduce
51
52 # Füge die DataFrames schrittweise zusammen
53 merged_df = reduce(lambda left, right: pd.merge(left, right, on=['LOR-Schlüssel', 'Bezirke'], how='outer', suffixes=('', '_dup')), filtered_dfs)
54
55 # Optional: Entfernen von doppelten Spalten, die durch die Zusammenführung
56 # entstehen könnten
57 merged_df = merged_df.loc[:, ~merged_df.columns.duplicated()]
58
59 # Sortieren nach 'Straftaten_insgesamt' absteigend
60 if 'Straftaten_insgesamt' in merged_df.columns:

```

```

58     sorted_df = merged_df.sort_values(by='Straftaten_insgesamt', ascending=
        False)
59 else:
60     print("'Straftaten_insgesamt' Spalte nicht gefunden. Sortierung ü
        bersprungen.")
61     sorted_df = merged_df
62
63 # Reset des Indexes
64 sorted_df.reset_index(drop=True, inplace=True)
65
66 # Ergebnis anzeigen
67 print(sorted_df)
68
69 # Optional: Speichern des sortierten DataFrames in eine neue Excel-Datei
70 sorted_df.to_excel('Zusammengefuegte_Fallzahlen.xlsx', index=False)
71 '''
72
73 ### Erklärung des Skripts:
74
75 1. **Importieren der Bibliotheken:**
76     - 'pandas' wird verwendet, um die Excel-Datei zu lesen und die Daten zu
        manipulieren.
77
78 2. **Festlegen des Pfads zur Excel-Datei:**
79     - Stelle sicher, dass sich die Datei 'Fallzahlen.xlsx' im selben
        Verzeichnis wie das Skript befindet oder gib den vollständigen Pfad an.
80
81 3. **Ausschließen der unerwünschten LOR-Schlüssel:**
82     - Die Liste 'exclude_lor_keys' enthält die LOR-Schlüssel, die nicht berü
        cksichtigt werden sollen (9999900 und 999999).
83
84 4. **Laden aller Sheets:**
85     - Mit 'sheet_name=None' werden alle Sheets als Dictionary geladen, wobei
        die Schlüssel die Sheet-Namen sind und die Werte die entsprechenden
        DataFrames.
86
87 5. **Filtern der Daten:**
88     - Für jedes Sheet wird geprüft, ob die notwendigen Spalten ('LOR-Schlü
        ssel' und 'Bezirke') vorhanden sind.
89     - Die unerwünschten LOR-Schlüssel werden ausgeschlossen.
90     - Optional kann eine Spalte hinzugefügt werden, die das ursprüngliche
        Sheet kennzeichnet.
91
92 6. **Zusammenführen der DataFrames:**
93     - Mit 'reduce' und 'pd.merge' werden alle gefilterten DataFrames auf Basis
        der Schlüssel 'LOR-Schlüssel' und 'Bezirke' zusammengeführt.
94     - Wenn es gleiche Spaltennamen in verschiedenen Sheets gibt, werden diese
        mit Suffixen versehen, um Duplikate zu vermeiden.
95
96 7. **Sortieren der Daten:**
97     - Der zusammengeführte DataFrame wird nach der Spalte '
        Straftaten_insgesamt' absteigend sortiert.

```

```

98 - Falls die Spalte nicht vorhanden ist, wird die Sortierung übersprungen.
99
100 8. **Speichern des Ergebnisses:**
101 - Der sortierte DataFrame wird angezeigt und optional in eine neue Excel-
    Datei namens 'Zusammengefuegte_Fallzahlen.xlsx' gespeichert.
102
103 ### Hinweise:
104
105 - **Überprüfung der Spaltennamen:** Stelle sicher, dass die Spaltennamen in
    allen Sheets konsistent sind. Groß- und Kleinschreibung sowie Leerzeichen
    können Einfluss auf die Funktionsweise des Skripts haben.
106
107 - **Datenkonsistenz:** Wenn verschiedene Sheets unterschiedliche Strukturen
    oder zusätzliche Spalten haben, kann dies zu unerwarteten Ergebnissen fö-
    hren. Passe das Skript ggf. an deine spezifischen Daten an.
108
109 - **Speicherleistung:** Bei sehr großen Excel-Dateien kann das Zusammenführen
    und Sortieren viel Speicher und Rechenleistung erfordern.
110
111 Ich hoffe, dieses Skript hilft dir weiter! Wenn du weitere Anpassungen benö-
    tigtst oder Fragen hast, lass es mich wissen.

```

Listing 88: AI Response Testfall 2 Prompt 2 Ausführung 3

```

1 Natürlich! Unten findest du ein Python-Skript, das deine Anforderungen erfü-
  llt. Dieses Skript liest alle Sheets der Excel-Datei 'Fallzahlen.xlsx',
  filtert die unerwünschten 'LOR-Schlüssel', fasst die Daten aller Bezirke
  zusammen, sortiert sie nach der akkumulierten Spalte 'Straftaten_insgesamt'
  und speichert das Ergebnis in einem Pandas DataFrame.
2
3 ### Voraussetzungen
4
5 Stelle sicher, dass du die erforderlichen Python-Pakete installiert hast. Du
  kannst sie mit 'pip' installieren, falls sie noch nicht vorhanden sind:
6
7 '''bash
8 pip install pandas openpyxl
9 '''
10
11 ### Python-Skript
12
13 '''python
14 import pandas as pd
15
16 def merge_and_sort_fallzahlen(excel_file):
17     """
18     Liest alle Sheets der angegebenen Excel-Datei, filtert unerwünschte LOR-
19     Schlüssel,
20     fasst die Daten der Bezirke zusammen, sortiert sie nach
21     Straftaten_insgesamt
22     und gibt das resultierende DataFrame zurück.
23
24     Parameters:

```

```

23     - excel_file (str): Pfad zur Excel-Datei (z.B. 'Fallzahlen.xlsx')
24
25     Returns:
26     - pd.DataFrame: Gefiltertes und sortiertes DataFrame
27     """
28
29     # Definiere die unerwünschten LOR-Schlüssel
30     exclude_lor_keys = [999900, 999999]
31
32     try:
33         # Lade alle Sheets in ein Dictionary von DataFrames
34         sheets_dict = pd.read_excel(excel_file, sheet_name=None)
35         print(f"Es wurden {len(sheets_dict)} Sheets gefunden und geladen.")
36     except FileNotFoundError:
37         print(f"Die Datei '{excel_file}' wurde nicht gefunden.")
38         return None
39     except Exception as e:
40         print(f"Ein Fehler ist beim Lesen der Excel-Datei aufgetreten: {e}")
41         return None
42
43     # Liste zur Speicherung gefilterter DataFrames
44     filtered_dfs = []
45
46     # Iteriere über jedes Sheet und filtere die unerwünschten Zeilen
47     for sheet_name, df in sheets_dict.items():
48         print(f"Verarbeite Sheet: {sheet_name}")
49
50         # Überprüfe, ob die erforderlichen Spalten vorhanden sind
51         required_columns = ['LOR-Schlüssel', 'Bezirke', 'Straftaten_insgesamt']
52
53         if not all(col in df.columns for col in required_columns):
54             print(f"Sheet '{sheet_name}' fehlt eine oder mehrere erforderliche Spalten. Überspringe dieses Sheet.")
55             continue
56
57         # Filtere die unerwünschten LOR-Schlüssel
58         df_filtered = df[~df['LOR-Schlüssel'].isin(exclude_lor_keys)].copy()
59         print(f"Anzahl der Bezirke nach Filtern in Sheet '{sheet_name}': {len(df_filtered)}")
60
61         # Optional: Falls du weitere Daten aggregieren möchtest, kannst du hier weitere Schritte hinzufügen
62
63         filtered_dfs.append(df_filtered)
64
65     if not filtered_dfs:
66         print("Keine gültigen Daten zum Zusammenfügen gefunden.")
67         return None
68
69     # Füge alle gefilterten DataFrames zusammen
70     combined_df = pd.concat(filtered_dfs, ignore_index=True)
71     print(f"Gesamtanzahl der Zeilen nach dem Zusammenfügen: {len(combined_df)}")

```

```

    }")
71
72     # Gruppieren die Daten nach 'LOR-Schlüssel' und 'Bezirke' und summiere '
Straftaten_insgesamt'
73     aggregated_df = combined_df.groupby(['LOR-Schlüssel', 'Bezirke'],
as_index=False)['Straftaten_insgesamt'].sum()
74     print("Daten wurden nach 'LOR-Schlüssel' und 'Bezirke' aggregiert.")
75
76     # Sortiere das DataFrame nach 'Straftaten_insgesamt' in absteigender
Reihenfolge
77     sorted_df = aggregated_df.sort_values(by='Straftaten_insgesamt',
ascending=False).reset_index(drop=True)
78     print("Daten wurden nach 'Straftaten_insgesamt' sortiert.")
79
80     return sorted_df
81
82 if __name__ == "__main__":
83     # Pfad zur Excel-Datei
84     excel_file = 'Fallzahlen.xlsx'
85
86     # Rufe die Funktion auf und erhalte das sortierte DataFrame
87     result_df = merge_and_sort_fallzahlen(excel_file)
88
89     if result_df is not None:
90         # Zeige die ersten paar Zeilen des Ergebnisses an
91         print("\nSortiertes Ergebnis:")
92         print(result_df.head())
93
94         # Optional: Speichere das Ergebnis in einer neuen Excel-Datei oder
CSV
95         # result_df.to_excel('Aggregierte_Fallzahlen.xlsx', index=False)
96         # result_df.to_csv('Aggregierte_Fallzahlen.csv', index=False)
97     '''
98
99     ### Erklärung der Schritte
100
101 1. **Importieren der Bibliotheken:**
102     - 'pandas' wird verwendet, um mit den Excel-Daten zu arbeiten.
103
104 2. **Definieren der Funktion 'merge_and_sort_fallzahlen':**
105     - Diese Funktion übernimmt den Pfad zur Excel-Datei, liest alle Sheets ein
, filtert die unerwünschten 'LOR-Schlüssel', fasst die Daten zusammen,
aggregiert die 'Straftaten_insgesamt' und sortiert das Ergebnis.
106
107 3. **Laden der Excel-Datei:**
108     - Mit 'pd.read_excel' und 'sheet_name=None' werden alle Sheets in ein
Dictionary geladen, wobei die Schlüssel die Sheet-Namen und die Werte die
entsprechenden DataFrames sind.
109
110 4. **Filtern der unerwünschten 'LOR-Schlüssel':**
111     - Für jedes Sheet werden die Zeilen mit 'LOR-Schlüssel' 999900 und 999999
entfernt, da diese nicht zu Bezirken gehören.

```



```

112
113 5. **Zusammenfügen der Daten:**
114 - Alle gefilterten DataFrames werden mit 'pd.concat' zu einem großen
    DataFrame kombiniert.
115
116 6. **Aggregieren der Daten:**
117 - Mit 'groupby' werden die Daten nach 'LOR-Schlüssel' und 'Bezirke'
    gruppiert, und die 'Straftaten_insgesamt' werden summiert.
118
119 7. **Sortieren des Ergebnisses:**
120 - Das aggregierte DataFrame wird nach 'Straftaten_insgesamt' in
    absteigender Reihenfolge sortiert.
121
122 8. **Ausführen des Skripts:**
123 - Wenn das Skript direkt ausgeführt wird, wird die Funktion aufgerufen,
    und das Ergebnis wird angezeigt. Optional kannst du das Ergebnis auch in
    eine neue Excel- oder CSV-Datei speichern, indem du die entsprechenden
    Zeilen entkommentierst.
124
125 ### Beispielausgabe
126
127 Nach dem Ausführen des Skripts solltest du eine Ausgabe ähnlich der folgenden
    erhalten (abhängig von deinen Daten):
128
129 '''
130 Es wurden 5 Sheets gefunden und geladen.
131 Verarbeite Sheet: Sheet1
132 Anzahl der Bezirke nach Filtern in Sheet 'Sheet1': 12
133 Verarbeite Sheet: Sheet2
134 Anzahl der Bezirke nach Filtern in Sheet 'Sheet2': 12
135 ...
136 Gesamtanzahl der Zeilen nach dem Zusammenfügen: 60
137 Daten wurden nach 'LOR-Schlüssel' und 'Bezirke' aggregiert.
138 Daten wurden nach 'Straftaten_insgesamt' sortiert.
139
140 Sortiertes Ergebnis:
141
142      LOR-Schlüssel      Bezirke  Straftaten_insgesamt
143 0      10000      Mitte      83611
144 1      20000  Friedrichshain-Kreuzberg      55838
145 2      80000      Neukölln      45313
146 3      30000      Pankow      39164
147 4      40000  Charlottenburg-Wilmersdorf      53630
148 ...
149 '''
150 ### Hinweise
151
152 - **Spaltennamen:** Stelle sicher, dass die Spaltennamen in deinem Excel-
    Dokument exakt mit den im Skript verwendeten Namen übereinstimmen.
    Andernfalls musst du die Spaltennamen im Skript entsprechend anpassen.
153
154 - **Weitere Aggregationen:** Falls du neben 'Straftaten_insgesamt' weitere

```

```

155     Spalten aggregieren möchtest, kannst du die 'groupby'-Funktion
156     entsprechend erweitern.
157
158 - **Fehlerbehandlung:** Das Skript enthält grundlegende Fehlerbehandlungen,
    um Probleme beim Laden der Datei oder fehlende Spalten zu erkennen.
159
160 - **Speichern der Ergebnisse:** Du kannst die aggregierten und sortierten
    Daten in eine neue Excel- oder CSV-Datei speichern, indem du die
    entsprechenden Zeilen am Ende des Skripts aktivierst.
161
162 Ich hoffe, dieses Skript hilft dir weiter! Bei weiteren Fragen stehe ich
    gerne zur Verfügung.

```

Listing 89: AI Response Testfall 2 Prompt 2 Ausführung 4

```

1 Natürlich! Hier ist ein Python-Skript, das die beschriebenen Anforderungen
  erfüllt. Es verwendet die 'pandas'-Bibliothek, um die Excel-Datei zu
  verarbeiten, die Daten aus allen Sheets zusammenzuführen, die nicht
  relevanten 'LOR-Schlüssel' zu entfernen, die Daten nach der akkumulierten
  Spalte 'Straftaten_insgesamt' pro Bezirk zu sortieren und das Ergebnis in
  einen 'DataFrame' zu speichern.
2
3 ### Schritt-für-Schritt-Anleitung:
4
5 1. **Importiere die notwendigen Bibliotheken**:
6     - 'pandas' für die Datenverarbeitung.
7     - 'openpyxl' oder 'xlrd' als Engine für das Lesen von Excel-Dateien (abhä
8       ngig von der Excel-Version).
9
10 2. **Lese alle Sheets der Excel-Datei ein**:
11     - Nutze 'pd.ExcelFile', um die Sheet-Namen zu erhalten.
12     - Iteriere über alle Sheets, lese sie ein und filtere die unerwünschten
13       Zeilen.
14
15 3. **Daten bereinigen und zusammenführen**:
16     - Entferne Zeilen mit den 'LOR-Schlüsseln' 999900 und 999999.
17     - Wähle relevante Spalten aus.
18     - Füge die Daten aus allen Sheets zusammen.
19
20 4. **Sortiere die Daten**:
21     - Gruppieren nach 'LOR-Schlüssel' und 'Bezirke'.
22     - Summiere die 'Straftaten_insgesamt' für jeden Bezirk.
23     - Sortiere die Bezirke basierend auf der summierten 'Straftaten_insgesamt
24       '-Spalte.
25
26 5. **Speichere das Ergebnis in einem DataFrame**.
27
28 ### Python-Skript:
29
30 '''python
31 import pandas as pd
32
33 def process_fallzahlen_excel(file_path):

```

```

31     """
32     Liest die Excel-Datei 'Fallzahlen.xlsx', verarbeitet die Daten aus allen
33     Sheets,
34     entfernt bestimmte LOR-Schlüssel, fasst die Daten zusammen, sortiert sie
35     nach
36     'Straftaten_insgesamt' und gibt den finalen DataFrame zurück.
37
38     :param file_path: Pfad zur Excel-Datei
39     :return: Gefilterter und sortierter Pandas DataFrame
40     """
41     # Lese die Excel-Datei
42     try:
43         xls = pd.ExcelFile(file_path)
44     except FileNotFoundError:
45         print(f"Die Datei {file_path} wurde nicht gefunden.")
46         return None
47     except Exception as e:
48         print(f"Ein Fehler ist aufgetreten: {e}")
49         return None
50
51     # Liste zur Speicherung der einzelnen DataFrames
52     df_list = []
53
54     # Iteriere über alle Sheets
55     for sheet_name in xls.sheet_names:
56         try:
57             df = pd.read_excel(xls, sheet_name=sheet_name)
58
59             # Überprüfe, ob die notwendigen Spalten vorhanden sind
60             required_columns = ['LOR-Schlüssel', 'Bezirke', '
61             Straftaten_insgesamt']
62             if not all(col in df.columns for col in required_columns):
63                 print(f"Ein oder mehrere erforderliche Spalten fehlen im
64                 Sheet '{sheet_name}'.")
65                 continue
66
67             # Entferne die Zeilen mit unerwünschten LOR-Schlüsseln
68             df_filtered = df[~df['LOR-Schlüssel'].isin([999900, 999999])]
69
70             # Wähle nur die benötigten Spalten
71             df_selected = df_filtered[required_columns]
72
73             # Füge eine neue Spalte für das Sheet hinzu (optional, falls benö
74             tigt)
75             df_selected['Sheet'] = sheet_name
76
77             # Füge den DataFrame der Liste hinzu
78             df_list.append(df_selected)
79
80         except Exception as e:
81             print(f"Fehler beim Verarbeiten des Sheets '{sheet_name}': {e}")
82             continue

```

```

78
79     if not df_list:
80         print("Keine Daten zum Verarbeiten gefunden.")
81         return None
82
83     # Füge alle DataFrames zusammen
84     combined_df = pd.concat(df_list, ignore_index=True)
85
86     # Gruppieren nach 'LOR-Schlüssel' und 'Bezirke' und summiere '
87     Straftaten_insgesamt'
88     grouped_df = combined_df.groupby(['LOR-Schlüssel', 'Bezirke'], as_index=
89     False)['Straftaten_insgesamt'].sum()
90
91     # Sortiere nach 'Straftaten_insgesamt' absteigend
92     sorted_df = grouped_df.sort_values(by='Straftaten_insgesamt', ascending=
93     False).reset_index(drop=True)
94
95     return sorted_df
96
97 if __name__ == "__main__":
98     # Pfad zur Excel-Datei
99     excel_file_path = 'Fallzahlen.xlsx'
100
101     # Verarbeite die Excel-Datei und erhalte den finalen DataFrame
102     final_df = process_fallzahlen_excel(excel_file_path)
103
104     if final_df is not None:
105         # Zeige die ersten paar Zeilen des finalen DataFrames an
106         print(final_df)
107
108         # Optional: Speichere den finalen DataFrame in einer neuen Excel-
109         oder CSV-Datei
110         # final_df.to_excel('Zusammengefasste_Fallzahlen.xlsx', index=False)
111         # final_df.to_csv('Zusammengefasste_Fallzahlen.csv', index=False)
112
113     """
114
115     ### Erklärung des Skripts:
116
117     1. **Funktion 'process_fallzahlen_excel':**
118         - **Parameter:** 'file_path' - der Pfad zur Excel-Datei.
119         - **Rückgabe:** Ein 'DataFrame', das die zusammengeführten und sortierten
120           Daten enthält.
121         - **Fehlerbehandlung:** Das Skript überprüft, ob die Datei existiert und
122           ob die erforderlichen Spalten in jedem Sheet vorhanden sind.
123
124     2. **Hauptteil des Skripts ('if __name__ == "__main__":')**
125         - Definiert den Pfad zur Excel-Datei.
126         - Ruft die Verarbeitungsfunktion auf.
127         - Gibt den finalen 'DataFrame' aus.
128         - Optional: Speichert das Ergebnis in einer neuen Excel- oder CSV-Datei.
129
130     ### Zusätzliche Hinweise:

```

```

124
125 - **Installation der benötigten Bibliotheken**:
126 Stelle sicher, dass du die 'pandas'-Bibliothek installiert hast. Falls
127 nicht, kannst du sie mit folgendem Befehl installieren:
128
129     '''bash
130     pip install pandas openpyxl
131     '''
132
133 Das Paket 'openpyxl' wird benötigt, um '.xlsx'-Dateien zu lesen.
134
135 - **Anpassungen**:
136 - Wenn du zusätzliche Spalten aus den Sheets benötigst, kannst du diese in
137   der Liste 'required_columns' hinzufügen.
138 - Die optionale Spalte 'Sheet' kann entfernt werden, wenn sie nicht benö-
139   tigt wird.
140
141 - **Ausgabe speichern**:
142 - Die letzten beiden Zeilen im Hauptteil des Skripts zeigen, wie du den
143   finalen 'DataFrame' in eine neue Datei speichern kannst. Du kannst diese
144   auskommentieren, falls du dies wünschst.
145
146 Dieses Skript bietet eine robuste Möglichkeit, die gewünschten Daten aus der
147 Excel-Datei zu extrahieren, zu verarbeiten und nach den Kriterien zu
148 sortieren, die du angegeben hast.

```

Listing 90: AI Response Testfall 2 Prompt 2 Ausführung 5

```

1 Hier ist ein Python-Skript, das die von Ihnen beschriebenen Anforderungen erf
  üllt. Das Skript verwendet die 'pandas'-Bibliothek, um die Excel-Datei zu
  lesen, die Daten zu verarbeiten und das sortierte DataFrame zurückzugeben.
2
3 Stellen Sie sicher, dass Sie die 'pandas'-Bibliothek installiert haben. Falls
  nicht, können Sie sie mit 'pip install pandas' installieren.
4
5     '''python
6     import pandas as pd
7
8     def main():
9         # 1. Excel-Datei einlesen und alle Sheets als DataFrames speichern
10        excel_datei = 'Fallzahlen.xlsx'
11        try:
12            # Liest alle Sheets und speichert sie in einem Dictionary
13            sheets_dict = pd.read_excel(excel_datei, sheet_name=None)
14        except FileNotFoundError:
15            print(f"Die Datei '{excel_datei}' wurde nicht gefunden.")
16            return
17        except Exception as e:
18            print(f"Fehler beim Lesen der Excel-Datei: {e}")
19            return
20
21        # 2. DataFrames der einzelnen Sheets zusammenführen
22        df_liste = []

```

```

23     for sheet_name, df in sheets_dict.items():
24         # Überprüfen, ob die erforderlichen Spalten existieren
25         if 'LOR-Schlüssel' not in df.columns or 'Bezirke' not in df.columns:
26             print(f"Sheet '{sheet_name}' enthält nicht die erforderlichen
27             Spalten.")
28             continue
29
30         # Setzen von 'LOR-Schlüssel' und 'Bezirke' als Index
31         df = df.set_index(['LOR-Schlüssel', 'Bezirke'])
32         df_liste.append(df)
33
34     if not df_liste:
35         print("Keine gültigen Sheets zum Verarbeiten gefunden.")
36         return
37
38     # Alle DataFrames zusammenfügen und numerische Spalten akkumulieren
39     zusammengefuegt_df = pd.concat(df_liste)
40     # Gruppieren nach den Index-Spalten und summieren der numerischen Werte
41     akkumuliert_df = zusammengefuegt_df.groupby(level=['LOR-Schlüssel', '
42     Bezirke']).sum().reset_index()
43
44     # 3. DataFrame sortieren nach 'Straftaten_insgesamt', spezielle Zeilen am
45     Ende platzieren
46     # Definieren der speziellen LOR-Schlüssel
47     spezielle_keys = [999900, 999999]
48
49     # Trennen der regulären und speziellen Zeilen
50     regulär_df = akkumuliert_df[~akkumuliert_df['LOR-Schlüssel'].isin(
51     spezielle_keys)]
52     spezielle_df = akkumuliert_df[akkumuliert_df['LOR-Schlüssel'].isin(
53     spezielle_keys)]
54
55     # Sortieren der regulären Zeilen nach 'Straftaten_insgesamt' absteigend
56     sortiert_regulär_df = regulär_df.sort_values(by='Straftaten_insgesamt',
57     ascending=False)
58
59     # Kombinieren der sortierten regulären Zeilen mit den speziellen Zeilen
60     endgültiges_df = pd.concat([sortiert_regulär_df, spezielle_df],
61     ignore_index=True)
62
63     # 4. Sortiertes DataFrame zurückgeben (hier als Ausgabe angezeigt)
64     print(endgültiges_df)
65
66 if __name__ == '__main__':
67     main()
68
69 """
70
71 ### Erklärung der Schritte:
72
73 1. **Excel-Datei einlesen**:
74     - Das Skript liest alle Sheets der Excel-Datei 'Fallzahlen.xlsx' ein und
75     speichert sie in einem Dictionary namens 'sheets_dict'.

```

```

67 - Es wird überprüft, ob die Datei existiert und ob alle notwendigen
    Spalten ('LOR-Schlüssel' und 'Bezirke') in jedem Sheet vorhanden sind.
68
69 2. **Daten zusammenführen**:
70 - Jedes DataFrame wird gesetzt mit 'LOR-Schlüssel' und 'Bezirke' als Index
    .
71 - Alle DataFrames werden mit 'pd.concat' zusammengefügt.
72 - Anschließend werden die Daten gruppiert und numerische Spalten werden
    summiert, um die akkumulierten Werte pro Bezirk zu erhalten.
73
74 3. **Sortieren des DataFrames**:
75 - Das DataFrame wird in reguläre Zeilen und spezielle Zeilen (mit 'LOR-
    Schlüssel' 999900 und 999999) aufgeteilt.
76 - Die regulären Zeilen werden nach der Spalte 'Straftaten_insgesamt'
    absteigend sortiert.
77 - Die speziellen Zeilen werden nach den regulären Zeilen angefügt, sodass
    sie am Ende des DataFrames stehen.
78
79 4. **Ausgabe des sortierten DataFrames**:
80 - Das sortierte DataFrame wird ausgegeben. Sie können es auch in eine neue
    Excel-Datei schreiben, indem Sie 'endgültiges_df.to_excel('
    sortierte_fallzahlen.xlsx', index=False)' verwenden.
81
82 ### Hinweise:
83 - **Fehlerbehandlung**: Das Skript umfasst grundlegende Fehlerbehandlungen, z
    .B. wenn die Datei nicht gefunden wird oder wenn ein Sheet die
    erforderlichen Spalten nicht enthält.
84 - **Anpassungen**: Falls die Struktur der Sheets unterschiedlich ist oder
    weitere Anpassungen erforderlich sind, müssen entsprechende Änderungen im
    Skript vorgenommen werden.
85 - **Ausgabe**: Zur Rückgabe des DataFrames wird 'print' verwendet. Falls Sie
    das Ergebnis weiter in einem Programm verwenden möchten, können Sie die
    Funktion 'main()' entsprechend anpassen, z.B. das DataFrame zurückgeben.
86
87 ### Beispielausführung:
88
89 Nachdem Sie das Skript gespeichert haben (z.B. als 'verarbeite_fallzahlen.py
    '), können Sie es in der Kommandozeile ausführen:
90
91 '''bash
92 python verarbeite_fallzahlen.py
93 '''
94
95 Dies gibt das sortierte DataFrame in der Konsole aus.

```

Listing 91: AI Response Testfall 2 Prompt 3 Ausführung 1

```

1 Hier ist ein Python-Skript, das die von Ihnen beschriebenen Anforderungen erf
    üllt. Das Skript verwendet die 'pandas' Bibliothek, um die Excel-Datei zu
    verarbeiten, die DataFrames zusammenzuführen, die Straftaten zu
    akkumulieren und das Ergebnis nach den gewünschten Kriterien zu sortieren.
2
3 Stellen Sie sicher, dass Sie die 'pandas' Bibliothek installiert haben. Falls

```

```

    nicht, können Sie sie mit 'pip' installieren:
4
5 '''bash
6 pip install pandas
7 '''
8
9 Hier ist das vollständige Skript:
10
11 '''python
12 import pandas as pd
13
14 def process_fallzahlen_excel(file_path='Fallzahlen.xlsx'):
15     """
16     Liest die Excel-Datei ein, führt die Sheets zusammen, akkumuliert die
17     Straftaten,
18     sortiert das DataFrame nach 'Straftaten_insgesamt' und platziert
19     spezielle
20     LOR-Schlüssel am Ende.
21
22     Parameters:
23     - file_path: str, der Pfad zur Excel-Datei.
24
25     Returns:
26     - pd.DataFrame: Das sortierte DataFrame.
27     """
28     # Schritt 1: Excel-Datei einlesen und alle Sheets als DataFrames
29     # speichern
30     try:
31         sheets = pd.read_excel(file_path, sheet_name=None)
32     except FileNotFoundError:
33         print(f"Die Datei {file_path} wurde nicht gefunden.")
34         return None
35     except Exception as e:
36         print(f"Fehler beim Einlesen der Datei: {e}")
37         return None
38
39     # Schritt 2: DataFrames zusammenführen und Straftaten akkumulieren
40     # Dabei werden alle Sheets kompiliert und numerische Werte summiert
41     try:
42         # Alle DataFrames zu einem einzigen DataFrame zusammenführen
43         df_all = pd.concat(sheets.values(), ignore_index=True)
44
45         # Gruppieren nach 'LOR-Schlüssel' und 'Bezirke' und numerische
46         # Spalten summieren
47         group_columns = ['LOR-Schlüssel', 'Bezirke']
48         df_grouped = df_all.groupby(group_columns, as_index=False).sum()
49     except KeyError as e:
50         print(f"Erforderliche Spalte fehlt: {e}")
51         return None
52     except Exception as e:
53         print(f"Fehler beim Zusammenführen der DataFrames: {e}")
54         return None

```



```

51
52     # Schritt 3: Sortieren nach 'Straftaten_insgesamt', spezielle LOR-Schlüssel am Ende
53     try:
54         # Definieren der speziellen LOR-Schlüssel
55         special_keys = [999900, 999999]
56
57         # Separieren der speziellen und normalen Zeilen
58         df_special = df_grouped[df_grouped['LOR-Schlüssel'].isin(special_keys)]
59         df_normal = df_grouped[~df_grouped['LOR-Schlüssel'].isin(special_keys)]
60
61         # Sortieren der normalen Zeilen nach 'Straftaten_insgesamt' absteigend
62         df_normal_sorted = df_normal.sort_values(by='Straftaten_insgesamt', ascending=False)
63
64         # Zusammenführen der sortierten normalen Zeilen mit den speziellen Zeilen
65         df_final = pd.concat([df_normal_sorted, df_special], ignore_index=True)
66     except KeyError:
67         print("Die Spalte 'Straftaten_insgesamt' ist im DataFrame nicht vorhanden.")
68         return None
69     except Exception as e:
70         print(f"Fehler beim Sortieren des DataFrames: {e}")
71         return None
72
73     # Schritt 4: Rückgabe des sortierten DataFrames
74     return df_final
75
76 if __name__ == "__main__":
77     # Verarbeiten der Excel-Datei und Erhalten des sortierten DataFrames
78     sorted_df = process_fallzahlen_excel('Fallzahlen.xlsx')
79
80     if sorted_df is not None:
81         # Anzeige der ersten paar Zeilen des sortierten DataFrames
82         print(sorted_df.head())
83
84         # Optional: Speichern des sortierten DataFrames in einer neuen Excel-Datei
85         sorted_df.to_excel('Fallzahlen_sortiert.xlsx', index=False)
86     '''
87
88     ### Erklärung der Schritte:
89
90     1. **Excel-Datei einlesen und Sheets als DataFrames speichern:**
91     - Die Funktion 'pd.read_excel' mit 'sheet_name=None' liest alle Sheets der Excel-Datei ein und speichert sie in einem Dictionary, wobei die Sheet-Namen die Schlüssel sind.

```

```

92
93 2. **DataFrames zusammenführen und Straftaten akkumulieren:**
94 - Alle einzelnen DataFrames werden mit 'pd.concat' zu einem großen
    DataFrame zusammengefügt.
95 - Anschließend wird das DataFrame nach den Spalten 'LOR-Schlüssel' und '
    Bezirke' gruppiert, und alle numerischen Spalten werden summiert, um die
    akkumulierten Werte der Straftaten zu erhalten.
96
97 3. **Sortieren nach 'Straftaten_insgesamt', spezielle LOR-Schlüssel am Ende
    :**
98 - Die Zeilen mit den LOR-Schlüsseln '999900' und '999999' werden separat
    gehalten.
99 - Die restlichen Zeilen werden nach der Spalte 'Straftaten_insgesamt' in
    absteigender Reihenfolge sortiert.
100 - Schließlich werden die speziellen Zeilen ans Ende des DataFrames angehä
    ngt.
101
102 4. **Rückgabe des sortierten DataFrames:**
103 - Das sortierte DataFrame wird zurückgegeben und kann weiter verwendet
    oder angezeigt werden.
104 - Optional können Sie das Ergebnis auch in eine neue Excel-Datei speichern
    , indem Sie die auskommentierte Zeile 'to_excel' verwenden.
105
106 ### Hinweise:
107
108 - **Fehlerbehandlung:** Das Skript enthält grundlegende Fehlerbehandlungen,
    um häufige Probleme wie fehlende Dateien oder Spalten abzufangen.
109 - **Anpassungen:** Je nach Struktur Ihrer Excel-Datei und den spezifischen
    Anforderungen könnten weitere Anpassungen erforderlich sein.
110 - **Speichern des Ergebnisses:** Falls Sie das Ergebnis in eine neue Datei
    speichern möchten, entfernen Sie den Kommentar vor der entsprechenden
    Zeile.
111
112 Ich hoffe, dieses Skript erfüllt Ihre Anforderungen. Bei weiteren Fragen oder
    Anpassungswünschen stehe ich gerne zur Verfügung!

```

Listing 92: AI Response Testfall 2 Prompt 3 Ausführung 2

```

1 '''python
2 import pandas as pd
3
4 def main():
5     # 1. Excel-Datei einlesen und alle Sheets als DataFrames speichern
6     excel_datei = 'Fallzahlen.xlsx'
7     sheets_dict = pd.read_excel(excel_datei, sheet_name=None)
8
9     # 2. DataFrames der einzelnen Sheets zusammenführen (joinen)
10    # Alle DataFrames vertikal concatenieren
11    concatenated_df = pd.concat(sheets_dict.values(), ignore_index=True)
12
13    # Gruppieren nach 'LOR-Schlüssel' und 'Bezirke' und die numerischen
    Spalten akkumulieren
14    gruppiert_df = concatenated_df.groupby(['LOR-Schlüssel', 'Bezirke'],

```

```

as_index=False).sum()

15
16 # 3. DataFrame sortieren nach 'Straftaten_insgesamt', dabei LOR-Schlüssel
17 999900 und 999999 am Ende lassen
18 # Definieren der auszuschließenden LOR-Schlüssel
19 ausschluss_keys = [999900, 999999]
20
21 # DataFrame ohne die auszuschließenden LOR-Schlüssel
22 zu_sortieren_df = gruppiert_df[~gruppiert_df['LOR-Schlüssel'].isin(
23 ausschluss_keys)]
24
25 # DataFrame mit den auszuschließenden LOR-Schlüsseln
26 ausgeschlossen_df = gruppiert_df[gruppiert_df['LOR-Schlüssel'].isin(
27 ausschluss_keys)]
28
29 # Sortieren nach 'Straftaten_insgesamt' in absteigender Reihenfolge
30 sortiert_df = zu_sortieren_df.sort_values(by='Straftaten_insgesamt',
31 ascending=False)
32
33 # Ausschließen Zeilen am Ende des sortierten DataFrames anhängen
34 final_df = pd.concat([sortiert_df, ausgeschlossen_df], ignore_index=True)
35
36 # 4. Sortiertes DataFrame zurückgeben (hier wird es angezeigt)
37 print(final_df)
38
39 if __name__ == '__main__':
40     main()
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99

```

### Erklärung der Schritte:

- 1. \*\*Excel-Datei einlesen:\*\***
  - Mit 'pd.read\_excel' wird die Excel-Datei 'Fallzahlen.xlsx' eingelesen.
  - 'sheet\_name=None' liest alle Sheets und speichert sie in einem Dictionary, wobei die Sheet-Namen die Schlüssel sind.
- 2. \*\*DataFrames zusammenführen:\*\***
  - Die einzelnen DataFrames werden mit 'pd.concat' vertikal zusammengeführt.
  - Anschließend wird das kombinierte DataFrame nach den Spalten 'LOR-Schlüssel' und 'Bezirke' gruppiert.
  - Mit '.sum()' werden die numerischen Spalten akkumuliert, sodass pro Bezirk die Gesamtwerte der Straftaten stehen.
- 3. \*\*Sortierung:\*\***
  - Definiert die LOR-Schlüssel, die nicht sortiert werden sollen ('999900' und '999999').
  - Trennt das DataFrame in zwei Teile: Einen Teil, der sortiert werden soll, und einen Teil, der am Ende bleiben soll.
  - Sortiert den ersten Teil nach 'Straftaten\_insgesamt' in absteigender Reihenfolge.
  - Fügt die ausgeschlossenen Zeilen am Ende des sortierten DataFrames hinzu

```

55
56 4. **Ausgabe:**
57 - Das final sortierte DataFrame wird mit 'print' ausgegeben. Je nach
    Bedarf kann dieses DataFrame auch in eine neue Excel-Datei oder ein
    anderes Format exportiert werden.
58
59 ### Zusätzliche Hinweise:
60
61 - **Installation der notwendigen Bibliothek:**
62 Stellen Sie sicher, dass 'pandas' installiert ist. Falls nicht, können Sie
    es mit folgendem Befehl installieren:
63     '''bash
64     pip install pandas
65     '''
66
67 - **Export des finalen DataFrames:**
68 Falls Sie das sortierte DataFrame in eine neue Excel-Datei speichern mö-
    chten, können Sie die folgende Zeile am Ende hinzufügen:
69     '''python
70     final_df.to_excel('Sortierte_Fallzahlen.xlsx', index=False)
71     '''
72
73 - **Fehlerbehandlung:**
74 Für eine robustere Anwendung könnten Sie zusätzliche Fehlerbehandlungen
    hinzufügen, z.B. Überprüfen, ob die erwarteten Spalten vorhanden sind.
75
76 Dieses Skript erfüllt die von Ihnen beschriebenen Anforderungen und sollte
    als Ausgangspunkt für weitere Anpassungen dienen, falls zusätzliche
    Funktionalitäten benötigt werden.

```

Listing 93: AI Response Testfall 2 Prompt 3 Ausführung 3

```

1 Hier ist ein Python-Skript, das die von dir beschriebenen Anforderungen erfü-
  llt. Das Skript verwendet die Bibliothek 'pandas', um die Excel-Datei zu
  verarbeiten. Stelle sicher, dass du 'pandas' und 'openpyxl' installiert
  hast. Du kannst sie mit 'pip' installieren, falls noch nicht geschehen:
2
3     '''bash
4     pip install pandas openpyxl
5     '''
6
7 ### Python-Skript: 'process_fallzahlen.py'
8
9     '''python
10    import pandas as pd
11
12    def process_fallzahlen(excel_file):
13        # 1. Die Excel-Datei einlesen und die Sheets als DataFrames speichern.
14        # Alle Sheets werden in einem Dictionary gespeichert, wobei der Sheet-
        # Name der Schlüssel ist.
15        xls = pd.ExcelFile(excel_file)
16        sheets_dict = {}

```

```

17     for sheet_name in xls.sheet_names:
18         df = pd.read_excel(xls, sheet_name=sheet_name)
19         sheets_dict[sheet_name] = df
20
21     # 2. Die DataFrames der einzelnen Sheets zusammen joinen,
22     # sodass pro Zeile der akkumulierte Wert der einzelnen Straftaten steht.
23     # Annahme: Alle DataFrames haben die selben 'LOR-Schlüssel' und 'Bezirke'
24     # Wir setzen 'LOR-Schlüssel' und 'Bezirke' als Index und summieren die
25     # numerischen Spalten.
26
27     # Liste der DataFrames
28     df_list = []
29     for df in sheets_dict.values():
30         # Setze 'LOR-Schlüssel' und 'Bezirke' als Index
31         df = df.set_index(['LOR-Schlüssel', 'Bezirke'])
32         df_list.append(df)
33
34     # Addiere alle DataFrames
35     # Dabei werden numerische Spalten summiert, nicht-numerische ignoriert
36     joined_df = pd.concat(df_list).groupby(['LOR-Schlüssel', 'Bezirke']).sum()
37     .reset_index()
38
39     # 3. Das neue gejointe DataFrame nach der Spalte "Straftaten_insgesamt"
40     # sortieren.
41     # Zeilen mit den LOR-Schlüsseln 999900 und 999999 werden nicht in die
42     # Sortierung einbezogen
43     # und am Ende des DataFrames angehängt.
44
45     # Trenne die speziellen LOR-Schlüssel
46     special_keys = [999900, 999999]
47     normal_df = joined_df[~joined_df['LOR-Schlüssel'].isin(special_keys)]
48     special_df = joined_df[joined_df['LOR-Schlüssel'].isin(special_keys)]
49
50     # Sortiere das normale DataFrame nach 'Straftaten_insgesamt' absteigend
51     sorted_normal_df = normal_df.sort_values(by='Straftaten_insgesamt',
52     ascending=False)
53
54     # Füge die speziellen Zeilen am Ende hinzu
55     final_df = pd.concat([sorted_normal_df, special_df], ignore_index=True)
56
57     # 4. Das sortierte Pandas DataFrame zurückgeben.
58     return final_df
59
60 if __name__ == "__main__":
61     # Pfad zur Excel-Datei
62     excel_file = 'Fallzahlen.xlsx'
63
64     # Verarbeite die Excel-Datei
65     sorted_df = process_fallzahlen(excel_file)
66
67     # Optional: Zeige die ersten Zeilen des sortierten DataFrames an

```

```

63     print(sorted_df.head())
64
65     # Optional: Speichere das sortierte DataFrame in eine neue Excel-Datei
66     sorted_df.to_excel('Fallzahlen_sortiert.xlsx', index=False)
67 '''
68
69 ### Erklärung der Schritte
70
71 1. **Excel-Datei einlesen und Sheets speichern:**
72     - Das Skript verwendet 'pd.ExcelFile', um alle Sheet-Namen der Excel-Datei
73       zu ermitteln.
74     - Jedes Sheet wird einzeln in ein DataFrame eingelesen und in einem
75       Dictionary 'sheets_dict' gespeichert.
76
77 2. **DataFrames zusammenführen:**
78     - Alle DataFrames werden so vorbereitet, dass 'LOR-Schlüssel' und '
79       Bezirke' als Index gesetzt werden.
80     - Die DataFrames werden mit 'pd.concat' zusammengeführt und anschließend
81       mit 'groupby' nach den Index-Spalten gruppiert.
82     - Die numerischen Spalten werden summiert, sodass pro Bezirk die
83       akkumulierten Werte der Straftaten vorliegen.
84
85 3. **Sortierung:**
86     - Die Zeilen mit den LOR-Schlüsseln '999900' und '999999' werden separat
87       behandelt und nach der Sortierung wieder ans Ende des DataFrames angehängt
88       .
89     - Das restliche DataFrame wird nach der Spalte 'Straftaten_insgesamt' in
90       absteigender Reihenfolge sortiert.
91
92 4. **Rückgabe und Speicherung:**
93     - Das sortierte DataFrame wird zurückgegeben.
94     - Im Hauptteil des Skripts ('if __name__ == "__main__":') wird das
95       Ergebnis optional angezeigt und in eine neue Excel-Datei '
96       Fallzahlen_sortiert.xlsx' gespeichert.
97
98 ### Nutzung
99
100 1. **Speichere das Skript:** Erstelle eine Python-Datei, z.B. '
101     process_fallzahlen.py', und füge den obigen Code ein.
102
103 2. **Führe das Skript aus:** Stelle sicher, dass sich die Excel-Datei '
104     Fallzahlen.xlsx' im selben Verzeichnis wie das Skript befindet oder passe
105     den Pfad entsprechend an. Führe dann das Skript aus:
106
107     ```bash
108     python process_fallzahlen.py
109     ```
110
111 3. **Ergebnis:** Das Skript zeigt die ersten Zeilen des sortierten DataFrames
112     an und speichert das gesamte sortierte DataFrame in der Datei '
113     Fallzahlen_sortiert.xlsx'.

```

```

100 ### Anmerkungen
101
102 - **Annahmen:** Das Skript geht davon aus, dass alle Sheets in der Excel-
    Datei ähnliche Strukturen haben, insbesondere die Spalten 'LOR-Schlüssel'
    ' und 'Bezirke'. Außerdem werden alle numerischen Spalten summiert;
    nicht-numerische Spalten außer 'LOR-Schlüssel' und 'Bezirke' werden
    ignoriert.
103
104 - **Fehlerbehandlung:** Für eine produktive Nutzung solltest du zusätzliche
    Fehlerbehandlungen einbauen, z.B. Überprüfung, ob die benötigten Spalten
    vorhanden sind, und Umgang mit fehlenden Werten.
105
106 - **Performance:** Bei sehr großen Excel-Dateien könnte die Performance eine
    Rolle spielen. In solchen Fällen könnten optimierte Lese- und
    Verarbeitungsmethoden notwendig sein.
107
108 Ich hoffe, dieses Skript erfüllt deine Anforderungen! Wenn du weitere
    Anpassungen benötigst, lass es mich wissen.

```

Listing 94: AI Response Testfall 2 Prompt 3 Ausführung 4

```

1 Hier ist ein vollständiges Python-Skript, das Ihre Anforderungen erfüllt. Das
  Skript verwendet die 'pandas'-Bibliothek, um die Excel-Datei zu
  verarbeiten. Stellen Sie sicher, dass Sie 'pandas' installiert haben ('pip
  install pandas'), bevor Sie das Skript ausführen.
2
3 '''python
4 import pandas as pd
5
6 def process_fallzahlen_excel(file_path='Fallzahlen.xlsx'):
7     """
8     Liest die Excel-Datei ein, verbindet die Sheets, sortiert das Ergebnis
9     nach
10    'Straftaten_insgesamt' und platziert die nicht zuordenbaren Einträge am
11    Ende.
12
13    Parameters:
14    - file_path: Pfad zur Excel-Datei (Standard: 'Fallzahlen.xlsx')
15
16    Returns:
17    - final_df: Sortiertes Pandas DataFrame
18    """
19    # Schritt 1: Alle Sheets der Excel-Datei einlesen
20    try:
21        excel_sheets = pd.read_excel(file_path, sheet_name=None)
22    except FileNotFoundError:
23        print(f"Datei '{file_path}' wurde nicht gefunden.")
24        return
25    except Exception as e:
26        print(f"Fehler beim Einlesen der Datei: {e}")
27        return
28
29    # Liste zur Speicherung der DataFrames

```

```

28     df_list = []
29
30     for sheet_name, df in excel_sheets.items():
31         # Sicherstellen, dass 'LOR-Schlüssel' und 'Bezirke' als Schlüssel
32         # vorhanden sind
33         if 'LOR-Schlüssel' not in df.columns or 'Bezirke' not in df.columns:
34             print(f"Sheet '{sheet_name}' enthält nicht die notwendigen
35             Spalten.")
36             continue
37         df_list.append(df)
38
39     if not df_list:
40         print("Keine gültigen Sheets gefunden.")
41         return
42
43     # Schritt 2: DataFrames zusammenführen (joinen) auf 'LOR-Schlüssel' und '
44     # Bezirke'
45     merged_df = df_list[0]
46     for df in df_list[1:]:
47         merged_df = pd.merge(merged_df, df, on=['LOR-Schlüssel', 'Bezirke'],
48                             how='outer', suffixes=('', '_dup'))
49
50         # Entfernen von doppelten Spalten, falls vorhanden
51         dup_columns = [col for col in merged_df.columns if col.endswith('_dup
52         ')]
53         if dup_columns:
54             merged_df.drop(columns=dup_columns, inplace=True)
55
56     # Schritt 3: Sortieren nach 'Straftaten_insgesamt', wobei 999900 und
57     # 999999 am Ende stehen
58     # Zuerst sicherstellen, dass 'Straftaten_insgesamt' numerisch ist
59     merged_df['Straftaten_insgesamt'] = pd.to_numeric(merged_df['
60     Straftaten_insgesamt'], errors='coerce')
61
62     # Trennen der DataFrames
63     districts_df = merged_df[~merged_df['LOR-Schlüssel'].isin([999900,
64     999999])]
65     non_districts_df = merged_df[merged_df['LOR-Schlüssel'].isin([999900,
66     999999])]
67
68     # Sortieren der Bezirke
69     sorted_districts_df = districts_df.sort_values(by='Straftaten_insgesamt',
70     ascending=False)
71
72     # Kombinieren der sortierten Bezirke mit den nicht zuordenbaren Einträgen
73     final_df = pd.concat([sorted_districts_df, non_districts_df],
74     ignore_index=True)
75
76     return final_df
77
78 if __name__ == "__main__":
79     # Pfad zur Excel-Datei

```



```

69     excel_file = 'Fallzahlen.xlsx'
70
71     # Verarbeitung der Excel-Datei
72     sorted_df = process_fallzahlen_excel(excel_file)
73
74     if sorted_df is not None:
75         # Ausgabe des sortierten DataFrames
76         print(sorted_df)
77
78         # Optional: Speichern des Ergebnisses in einer neuen Excel-Datei
79         # sorted_df.to_excel('Sorted_Fallzahlen.xlsx', index=False)
80     '''
81
82     ### Erklärung der Schritte:
83
84     1. **Einlesen der Excel-Datei:**
85     - Das Skript liest alle Sheets der angegebenen Excel-Datei mit 'pd.
86       read_excel' ein. Jeder Sheet wird als separates DataFrame gespeichert.
87     - Es wird überprüft, ob die notwendigen Spalten 'LOR-Schlüssel' und '
88       Bezirke' in jedem Sheet vorhanden sind.
89
90     2. **Zusammenführen der DataFrames:**
91     - Die DataFrames werden sequenziell auf den Schlüsselspalten 'LOR-Schlü
92       ssel' und 'Bezirke' mit einem äußeren Join ('how='outer') zusammengefü
93       hrt.
94     - Doppelte Spalten, die durch den Merge entstehen könnten (z.B. gleiche
95       Spalten aus verschiedenen Sheets), werden entfernt.
96
97     3. **Sortieren des DataFrames:**
98     - Das Skript trennt die Bezirke (mit gültigen 'LOR-Schlüssel') von den
99       nicht zuordenbaren Einträgen ('999900' und '999999').
100    - Die Bezirke werden absteigend nach 'Straftaten_insgesamt' sortiert.
101    - Die nicht zuordenbaren Einträge werden anschließend an das Ende des
102      DataFrames angehängt.
103
104    4. **Ausgabe:**
105    - Das sortierte DataFrame wird ausgegeben. Optional kann es auch in eine
106      neue Excel-Datei gespeichert werden, indem die Kommentarzeile am Ende des
107      Skripts aktiviert wird.
108
109    ### Hinweise:
110
111    - **Fehlerbehandlung:** Das Skript enthält grundlegende Fehlerbehandlungen,
112      um sicherzustellen, dass die Datei existiert und die notwendigen Spalten
113      in den Sheets vorhanden sind.
114    - **Anpassungen:** Falls die Struktur Ihrer Excel-Sheets anders ist oder zusä
115      tzliche Verarbeitungsschritte erforderlich sind, passen Sie das Skript
116      entsprechend an.
117    - **Leistung:** Bei sehr großen Excel-Dateien könnte die Leistung beeinträ
118      chtigt werden. In solchen Fällen können Optimierungen notwendig sein.
119
120    Stellen Sie sicher, dass die Excel-Datei 'Fallzahlen.xlsx' im gleichen

```

Verzeichnis wie das Skript liegt oder geben Sie den vollständigen Pfad zur Datei an.

Listing 95: AI Response Testfall 2 Prompt 3 Ausführung 5

```
1 Natürlich! Hier ist ein Python-Skript, das die gewünschten Berechnungen
  durchführt. Dieses Skript verwendet die 'pandas' Bibliothek, um die Excel-
  Datei zu lesen, die prozentualen Anteile der Straftaten pro Bezirk zu
  berechnen und die Ergebnisse in einer neuen Excel-Datei zu speichern.
2
3 ### Voraussetzungen
4
5 Stelle sicher, dass du die 'pandas' und 'openpyxl' Bibliotheken installiert
  hast. Falls nicht, kannst du sie mit folgendem Befehl installieren:
6
7 ```bash
8 pip install pandas openpyxl
9 ```
10
11 ### Python-Skript
12
13 ```python
14 import pandas as pd
15
16 def berechne_straftaten_anteil(excel_datei, sheet_name, ausgabe_datei):
17     # Excel-Datei einlesen
18     df = pd.read_excel(excel_datei, sheet_name=sheet_name)
19
20     # Anzeigen der Daten zur Überprüfung (optional)
21     # print(df.head())
22
23     # Identifizieren der Gesamtzeile für Berlin
24     # Annahme: Die Gesamtzeile hat 'Bezirke' als "Berlin (PKS gesamt)"
25     gesamt_row = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
26
27     if gesamt_row.empty:
28         # Falls keine explizite Gesamtzeile vorhanden ist, summiere alle
29         # relevanten Bezirke
30         # Annahme: Die Gesamtzeile hat einen LOR-Schlüssel von 999999
31         gesamt_row = df[df['LOR-Schlüssel'] == 999999]
32
33     if gesamt_row.empty:
34         raise ValueError("Gesamtdaten für Berlin nicht gefunden.")
35
36     # Gesamtzahl der Straftaten in Berlin
37     gesamt_straftaten = gesamt_row['Straftaten_insgesamt'].values[0]
38
39     # Filtere nur die Bezirke, die zu Berlin gehören und nicht die
40     # Gesamtzeile oder nicht zuzuordnend
41     # Annahme: 'Stadtgebiet Berlin, nicht zuzuordnen' und 'Berlin (PKS gesamt)'
42     # sollen ausgeschlossen werden
43     relevante_bezirke = df[~df['Bezirke'].isin(['Berlin (PKS gesamt)', '
44     Stadtgebiet Berlin, nicht zuzuordnen'])]
```

```

41
42 # Berechne den prozentualen Anteil
43 relevante_bezirke = relevante_bezirke.copy() # Um eine Kopie zu
erstellen
44 relevante_bezirke['Prozentualer Anteil (%)'] = (relevante_bezirke['
Straftaten_insgesamt'] / gesamt_straftaten) * 100
45
46 # Optional: Runden auf zwei Dezimalstellen
47 relevante_bezirke['Prozentualer Anteil (%)'] = relevante_bezirke['
Prozentualer Anteil (%)'].round(2)
48
49 # Sortiere die Bezirke nach dem prozentualen Anteil absteigend
50 relevante_bezirke = relevante_bezirke.sort_values(by='Prozentualer Anteil
(%)', ascending=False)
51
52 # Auswahl der relevanten Spalten für die Ausgabe
53 ausgabe_df = relevante_bezirke[['Bezirke', 'Straftaten_insgesamt', '
Prozentualer Anteil (%)']]
54
55 # Ausgabe in eine neue Excel-Datei
56 ausgabe_df.to_excel(ausgabe_datei, index=False)
57
58 print(f"Die prozentualen Anteile wurden erfolgreich berechnet und in '{
ausgabe_datei}' gespeichert.")
59
60 if __name__ == "__main__":
61     # Pfad zur Eingabe-Excel-Datei
62     input_excel = 'Fallzahlen.xlsx'
63
64     # Name des Sheets
65     sheet = 'Fallzahlen_2023'
66
67     # Name der Ausgabedatei
68     output_excel = 'Straftaten_Anteil_Bezirke.xlsx'
69
70     # Funktion aufrufen
71     berechne_straftaten_anteil(input_excel, sheet, output_excel)
72     '''
73
74     ### Erklärung des Skripts
75
76     1. **Importieren der Bibliothek**:
77         - 'pandas' wird verwendet, um die Excel-Datei zu lesen und die Daten zu
verarbeiten.
78
79     2. **Funktion 'berechne_straftaten_anteil'**:
80         - **Einlesen der Excel-Datei**: Die Funktion liest die angegebene Excel-
Datei und das spezifische Sheet ein.
81         - **Identifizieren der Gesamtzeile**: Es wird nach der Zeile gesucht, die
die Gesamtsumme der Straftaten für Berlin enthält. Falls diese nicht
explizit vorhanden ist, wird angenommen, dass der LOR-Schlüssel '999999'
die Gesamtsumme darstellt.

```

```

82     - **Berechnung des prozentualen Anteils**: Für jeden Bezirk wird der
      Anteil der Straftaten an der Gesamtsumme berechnet.
83     - **Sortierung und Auswahl**: Die Bezirke werden nach ihrem prozentualen
      Anteil sortiert und relevante Spalten für die Ausgabe ausgewählt.
84     - **Speichern der Ergebnisse**: Die berechneten Anteile werden in einer
      neuen Excel-Datei gespeichert.
85
86 3. **Ausführung des Skripts**:
87     - Das Skript wird nur ausgeführt, wenn es direkt gestartet wird ('if
      __name__ == "__main__":').
88     - Du kannst die Variablen 'input_excel', 'sheet' und 'output_excel' nach
      Bedarf anpassen.
89
90 ### Ergebnis
91
92 Nach dem Ausführen des Skripts findest du eine neue Excel-Datei namens '
      Straftaten_Anteil_Bezirke.xlsx' im gleichen Verzeichnis, in dem das Skript
      ausgeführt wurde. Diese Datei enthält die Bezirke, die Gesamtzahl der
      Straftaten pro Bezirk und den prozentualen Anteil jedes Bezirks an den
      gesamten Straftaten in Berlin.
93
94 ### Beispielausgabe
95
96 | Bezirke                | Straftaten_insgesamt | Prozentualer Anteil
97 | -----|-----|-----|
98 | Mitte                  | 83,611                | 15.58
99 |                         |                       |
100 | Neukölln               | 45,313                | 8.47
101 |                         |                       |
102 | Friedrichshain-Kreuzberg | 55,838                | 10.43
103 | ...                    | ...                   | ...
104 |                         |                       |
105
106 Dieses Format ermöglicht eine klare und übersichtliche Darstellung der
      prozentualen Anteile der Straftaten pro Bezirk in Berlin.

```

Listing 96: AI Response Testfall 3 Prompt 1 Ausführung 1

```

1 Natürlich! Hier ist ein Python-Skript, das die gewünschten Berechnungen
  durchführt. Es verwendet die Bibliothek 'pandas', um die Excel-Datei zu
  lesen und die prozentualen Anteile der Straftaten pro Bezirk im Verhältnis
  zu den gesamten Straftaten in Berlin zu berechnen.
2
3 Stellen Sie sicher, dass Sie die benötigten Bibliotheken installiert haben.
  Wenn nicht, können Sie sie mit 'pip' installieren:
4
5 '''bash
6 pip install pandas openpyxl
7 '''
8

```

```

9 Hier ist das vollständige Skript:
10
11 '''python
12 import pandas as pd
13
14 # Pfad zur Excel-Datei
15 excel_file = 'Fallzahlen.xlsx'
16
17 # Name des Sheets
18 sheet_name = 'Fallzahlen_2023'
19
20 # Lesen des Excel-Sheets
21 df = pd.read_excel(excel_file, sheet_name=sheet_name)
22
23 # Annehmen, dass die Gesamtzahl in der Zeile mit 'Berlin (PKS gesamt)' steht
24 # Suchen der Gesamtzeile
25 total_row = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
26
27 if total_row.empty:
28     raise ValueError("Die Gesamtzeile 'Berlin (PKS gesamt)' wurde nicht
29         gefunden.")
30
31 # Extrahieren der Gesamtzahl der Straftaten
32 total_crimes = total_row['Straftaten_insgesamt'].values[0]
33
34 # Filtern der Bezirke (ohne Gesamtzeile)
35 districts_df = df[df['Bezirke'] != 'Berlin (PKS gesamt)'].copy()
36
37 # Berechnen des prozentualen Anteils
38 districts_df['Prozentualer_Anteil (%)'] = (districts_df['Straftaten_insgesamt']
39     / total_crimes) * 100
40
41 # Optional: Runden auf zwei Dezimalstellen
42 districts_df['Prozentualer_Anteil (%)'] = districts_df['Prozentualer_Anteil
43     (%)'].round(2)
44
45 # Anzeigen der Ergebnisse
46 print(districts_df[['Bezirke', 'Straftaten_insgesamt', 'Prozentualer_Anteil
47     (%)']])
48
49 # Optional: Speichern der Ergebnisse in eine neue Excel-Datei
50 output_file = 'Straftaten_Prozentual_Anteil.xlsx'
51 districts_df.to_excel(output_file, sheet_name='Prozentuale Anteile', index=
52     False)
53 print(f"\nDie Ergebnisse wurden in '{output_file}' gespeichert.")
54 '''
55
56 ### Erklärung des Skripts:
57
58 1. **Importieren der Bibliotheken:**
59     - 'pandas' wird verwendet, um die Excel-Datei zu lesen und zu manipulieren
60     .

```

```

55
56 2. **Lesen der Excel-Datei:**
57 - Das Skript liest das Sheet 'Fallzahlen_2023' aus der Datei '
    Fallzahlen.xlsx'.
58
59 3. **Identifizieren der Gesamtzeile:**
60 - Es wird angenommen, dass die Gesamtzahl der Straftaten in der Zeile mit
    'Bezirke' als 'Berlin (PKS gesamt)' steht.
61 - Falls diese Zeile nicht gefunden wird, wirft das Skript einen Fehler.
62
63 4. **Berechnung des prozentualen Anteils:**
64 - Das Skript filtert die Gesamtzeile heraus und berechnet für jeden Bezirk
    den prozentualen Anteil der Straftaten im Vergleich zur Gesamtzahl.
65 - Die Ergebnisse werden auf zwei Dezimalstellen gerundet.
66
67 5. **Ausgabe und Speicherung:**
68 - Die Ergebnisse werden in der Konsole angezeigt.
69 - Optional speichert das Skript die Ergebnisse in eine neue Excel-Datei
    namens 'Straftaten_Prozentual_Anteil.xlsx'.
70
71 ### Beispielausgabe:
72
73 '''
74
75         Bezirke  Straftaten_insgesamt  Prozentual_Anteil (%)
76 0             Mitte                83611                15.58
77 1  Friedrichshain-Kreuzberg                55838
78    12.39
79 2             Pankow                39164                7.31
80 3  Charlottenburg-Wilmersdorf                53630                10.00
81 4             Spandau                25564                4.77
82 5  Steglitz-Zehlendorf                23719                4.42
83 6  Tempelhof-Schöneberg                41776                7.79
84 7             Neukölln                45313                8.46
85 8  Treptow-Köpenick                26099                4.87
86 9  Marzahn-Hellersdorf                26649                4.98
87 10             Lichtenberg                31563                5.89
88 11             Reinickendorf                27344                5.10
89 12 Stadtgebiet Berlin, nicht zuzuordnen                56425
90    10.53
91
92 Die Ergebnisse wurden in 'Straftaten_Prozentual_Anteil.xlsx' gespeichert.
93 '''
94
95 ### Hinweise:
96
97 - **Dateipfade:** Stellen Sie sicher, dass sich die Datei 'Fallzahlen.xlsx'
    im gleichen Verzeichnis wie das Skript befindet oder geben Sie den vollst
    ändigen Pfad zur Datei an.
98
99 - **Fehlerbehandlung:** Das Skript überprüft, ob die Gesamtzeile vorhanden
    ist. Sie können zusätzliche Fehlerprüfungen hinzufügen, je nach Bedarf.

```

```

98 - **Anpassungen:** Sie können das Skript erweitern, um weitere Analysen
    durchzuführen oder die Ausgabe nach Ihren Wünschen anzupassen.
99
100 Ich hoffe, dieses Skript hilft Ihnen weiter! Wenn Sie weitere Anpassungen
    oder Fragen haben, lassen Sie es mich wissen.

```

Listing 97: AI Response Testfall 3 Prompt 1 Ausführung 2

```

1 Hier ist ein Python-Skript, das den prozentualen Anteil der gesamten
  Straftaten der einzelnen Bezirke von den gesamten Straftaten in ganz
  Berlin berechnet. Das Skript verwendet die Bibliothek 'pandas', um die
  Excel-Datei zu lesen und die Berechnungen durchzuführen.
2
3 ### Voraussetzungen
4
5 Stellen Sie sicher, dass die 'pandas'-Bibliothek installiert ist. Falls nicht
  , können Sie sie mit dem folgenden Befehl installieren:
6
7 '''bash
8 pip install pandas openpyxl
9 '''
10
11 ### Python-Skript
12
13 '''python
14 import pandas as pd
15
16 # Pfad zur Excel-Datei
17 excel_datei = 'Fallzahlen.xlsx'
18
19 # Name des Sheets
20 sheet_name = 'Fallzahlen_2023'
21
22 # Lesen der Excel-Datei
23 try:
24     df = pd.read_excel(excel_datei, sheet_name=sheet_name)
25 except FileNotFoundError:
26     print(f"Die Datei '{excel_datei}' wurde nicht gefunden.")
27     exit(1)
28 except Exception as e:
29     print(f"Fehler beim Lesen der Excel-Datei: {e}")
30     exit(1)
31
32 # Anzeigen der geladenen Daten (optional)
33 #print(df.head())
34
35 # Herausfinden der Gesamtstraftaten in Berlin
36 # Annahme: Die Gesamtstraftaten stehen in der Zeile mit 'Berlin (PKS gesamt)'
37 gesamt_row = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
38
39 if gesamt_row.empty:
40     print("Die Zeile 'Berlin (PKS gesamt)' wurde nicht gefunden.")
41     exit(1)

```

```

42
43 gesamt_straftaten = gesamt_row['Straftaten_insgesamt'].values[0]
44
45 # Filtern der Bezirke (Ausschluss von Gesamtzeilen)
46 bezirke_df = df[~df['Bezirke'].isin(['Berlin (PKS gesamt)', 'Stadtgebiet
    Berlin, nicht zuzuordnen'])]
47
48 # Berechnung des prozentualen Anteils
49 bezirke_df = bezirke_df.copy() # Vermeidung von SettingWithCopyWarning
50 bezirke_df['Prozentualer_Anteil (%)'] = (bezirke_df['Straftaten_insgesamt'] /
    gesamt_straftaten) * 100
51
52 # Optional: Runden auf zwei Dezimalstellen
53 bezirke_df['Prozentualer_Anteil (%)'] = bezirke_df['Prozentualer_Anteil (%)'
    ].round(2)
54
55 # Sortieren nach Prozentualem Anteil absteigend (optional)
56 bezirke_df = bezirke_df.sort_values(by='Prozentualer_Anteil (%)', ascending=
    False)
57
58 # Ergebnis anzeigen
59 ergebnis = bezirke_df[['Bezirke', 'Straftaten_insgesamt', '
    Prozentualer_Anteil (%)']]
60 print(ergebnis)
61
62 # Optional: Ergebnis in eine neue Excel-Datei speichern
63 # ergebnis.to_excel('Prozentuale_Anteile_Straftaten.xlsx', index=False)
64 '''
65
66 ### Erklärung des Skripts
67
68 1. **Importieren der Bibliothek:**
69     - 'pandas' wird zum Lesen und Verarbeiten der Excel-Datei verwendet.
70
71 2. **Lesen der Excel-Datei:**
72     - Das Skript versucht, die angegebene Excel-Datei und das spezifizierte
        Sheet zu lesen. Fehler beim Lesen der Datei werden abgefangen und
        ausgegeben.
73
74 3. **Bestimmen der Gesamtstrafataten:**
75     - Die Gesamtanzahl der Straftaten in Berlin wird aus der Zeile extrahiert,
        die den Bezirk 'Berlin (PKS gesamt)' enthält.
76
77 4. **Filtern der Bezirke:**
78     - Bezirke wie 'Stadtgebiet Berlin, nicht zuzuordnen' und 'Berlin (PKS
        gesamt)' werden aus der Berechnung ausgeschlossen.
79
80 5. **Berechnung des prozentualen Anteils:**
81     - Für jeden verbleibenden Bezirk wird der prozentuale Anteil der
        Straftaten im Vergleich zur Gesamtzahl berechnet.
82     - Der Anteil wird auf zwei Dezimalstellen gerundet.
83

```



```

84 6. **Sortierung und Ausgabe:**
85     - Die Bezirke werden nach dem prozentualen Anteil absteigend sortiert.
86     - Das Ergebnis wird angezeigt und optional in eine neue Excel-Datei
      gespeichert.
87
88 ### Ausgabe
89
90 Die Ausgabe des Skripts sieht ungefähr so aus:
91
92 '''
93             Bezirke  Straftaten_insgesamt  Prozentualer_Anteil (%)
94 0             Mitte                83611                xx.xx
95 1  Friedrichshain-Kreuzberg                55838                xx.
      xx
96 2             Pankow                39164                xx.xx
97 ...             ...                ...                ...
98 '''
99
100 Jeder Bezirk wird zusammen mit der Gesamtanzahl der Straftaten und dem
      berechneten prozentualen Anteil angezeigt.
101
102 ### Speicherung der Ergebnisse
103
104 Falls Sie die Ergebnisse in eine neue Excel-Datei speichern möchten,
      entkommentieren Sie die entsprechende Zeile am Ende des Skripts:
105
106 '''python
107 ergebnis.to_excel('Prozentuale_Anteile_Straftaten.xlsx', index=False)
108 '''
109
110 Dies erstellt eine Datei namens 'Prozentuale_Anteile_Straftaten.xlsx' im
      selben Verzeichnis wie das Skript.
111
112 ---
113
114 Passen Sie bei Bedarf den Pfad zur Excel-Datei und den Namen des Sheets an.
      Stellen Sie sicher, dass die Spaltennamen im Skript mit denen in Ihrer
      Excel-Datei übereinstimmen.

```

Listing 98: AI Response Testfall 3 Prompt 1 Ausführung 3

```

1 Hier ist ein Python-Skript, das die prozentualen Anteile der gesamten
      Straftaten der einzelnen Bezirke im Vergleich zu den gesamten Straftaten
      in ganz Berlin berechnet. Das Skript verwendet die 'pandas'-Bibliothek, um
      die Excel-Datei zu verarbeiten.
2
3 Stellen Sie sicher, dass Sie 'pandas' und 'openpyxl' installiert haben. Sie k
      önnen diese Pakete mit folgendem Befehl installieren, falls sie noch nicht
      installiert sind:
4
5 '''bash
6 pip install pandas openpyxl
7 '''

```

```

8
9  ### Python-Skript: 'berechne_prozentuale_anteile.py'
10
11  '''python
12  import pandas as pd
13
14  def berechne_prozentuale_anteile(dateipfad: str, sheet_name: str = '
    Fallzahlen_2023'):
15      """
16      Berechnet den prozentualen Anteil der Straftaten jedes Bezirks an den
    gesamten Straftaten in Berlin.
17
18      :param dateipfad: Pfad zur Excel-Datei 'Fallzahlen.xlsx'.
19      :param sheet_name: Name des Sheets in der Excel-Datei. Standard ist '
    Fallzahlen_2023'.
20      :return: DataFrame mit Bezirken und deren prozentualen Anteilen.
21      """
22      try:
23          # Excel-Datei lesen
24          df = pd.read_excel(dateipfad, sheet_name=sheet_name)
25      except FileNotFoundError:
26          print(f"Die Datei {dateipfad} wurde nicht gefunden.")
27          return
28      except Exception as e:
29          print(f"Ein Fehler ist beim Lesen der Excel-Datei aufgetreten: {e}")
30          return
31
32      # Gesamtstraftaten für Berlin abrufen
33      total_crimes_row = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
34      if not total_crimes_row.empty:
35          total_crimes = total_crimes_row['Straftaten_insgesamt'].values[0]
36      else:
37          # Falls 'Berlin (PKS gesamt)' nicht vorhanden ist, summiere die
    Bezirke
38          exclude_bezirke = ['Berlin (PKS gesamt)', 'Stadtgebiet Berlin, nicht
    zuzuordnen']
39          total_crimes = df.loc[~df['Bezirke'].isin(exclude_bezirke), '
    Straftaten_insgesamt'].sum()
40
41      # Bezirke filtern (ausschließen: Gesamtdaten und nicht zuordenbare
    Gebiete)
42      exclude_bezirke = ['Berlin (PKS gesamt)', 'Stadtgebiet Berlin, nicht
    zuzuordnen']
43      bezirke_df = df.loc[~df['Bezirke'].isin(exclude_bezirke)].copy()
44
45      # Prozentualen Anteil berechnen
46      bezirke_df['Prozentualer_Anteil'] = (bezirke_df['Straftaten_insgesamt'] /
    total_crimes) * 100
47
48      # Optional: Auf zwei Dezimalstellen runden
49      bezirke_df['Prozentualer_Anteil'] = bezirke_df['Prozentualer_Anteil'].
    round(2)

```

```

50
51 # Ergebnis anzeigen
52 print("Prozentualer Anteil der Straftaten nach Bezirk:")
53 print(bezirke_df[['Bezirke', 'Straftaten_insgesamt', 'Prozentualer_Anteil',
54 '']])
55
56 # Optional: Ergebnis in eine neue Excel-Datei speichern
57 output_datei = 'Prozentuale_Anteile_Straftaten.xlsx'
58 try:
59     bezirke_df[['Bezirke', 'Straftaten_insgesamt', 'Prozentualer_Anteil',
60 '']].to_excel(output_datei, index=False)
61     print(f"\nDie Ergebnisse wurden in '{output_datei}' gespeichert.")
62 except Exception as e:
63     print(f"Ein Fehler ist beim Speichern der Ergebnisdatei aufgetreten: {e}")
64
65 if __name__ == "__main__":
66     # Pfad zur Excel-Datei
67     dateipfad = 'Fallzahlen.xlsx'
68
69     # Prozentuale Anteile berechnen
70     berechne_prozentuale_anteile(dateipfad)
71
72
73 ### Erklärung des Skripts
74
75 1. **Importieren der Bibliothek:**
76 - 'pandas' wird verwendet, um die Excel-Datei zu lesen und die Daten zu
77 verarbeiten.
78
79 2. **Funktion 'berechne_prozentuale_anteile':**
80 - **Parameter:**
81 - 'dateipfad': Der Pfad zur Excel-Datei 'Fallzahlen.xlsx'.
82 - 'sheet_name': Der Name des Sheets, standardmäßig auf 'Fallzahlen_2023'
83 gesetzt.
84
85 - **Schritte:**
86 1. **Lesen der Excel-Datei:**
87 - Versucht die Excel-Datei zu lesen. Falls die Datei nicht gefunden
88 wird oder ein anderer Fehler auftritt, wird eine Fehlermeldung ausgegeben.
89
90 2. **Bestimmen der Gesamtanzahl der Straftaten:**
91 - Sucht nach der Zeile, die den Gesamtwert für Berlin enthält ('Bezirke' == 'Berlin (PKS gesamt)').
92 - Falls diese Zeile nicht gefunden wird, summiert das Skript die
93 Straftaten aller Bezirke, außer denen, die ausgeschlossen werden sollen.
94
95 3. **Filtern der Bezirke:**
96 - Entfernt Zeilen, die nicht einem spezifischen Bezirk zugeordnet
97 werden können ('Stadtgebiet Berlin, nicht zuzuordnen' und 'Berlin (PKS
98 gesamt)').
99
100

```

```

92     4. **Berechnung des prozentualen Anteils:**
93         - Für jeden Bezirk wird der Anteil der Straftaten an den gesamten
          Straftaten Berlins berechnet.
94         - Das Ergebnis wird auf zwei Dezimalstellen gerundet.
95
96     5. **Ausgabe und Speicherung:**
97         - Das Ergebnis wird in der Konsole angezeigt.
98         - Optional wird das Ergebnis in eine neue Excel-Datei '
          Prozentuale_Anteile_Straftaten.xlsx' gespeichert.
99
100 3. **Ausführung des Skripts:**
101     - Das Skript wird ausgeführt, wenn es direkt aufgerufen wird.
102     - Der Pfad zur Excel-Datei wird festgelegt und die Funktion zur Berechnung
          der prozentualen Anteile wird aufgerufen.
103
104 ### Verwendung
105
106 1. **Platzieren Sie die Excel-Datei:**
107     - Stellen Sie sicher, dass sich die Datei 'Fallzahlen.xlsx' im gleichen
          Verzeichnis wie das Skript befindet oder geben Sie den vollständigen Pfad
          zur Datei an.
108
109 2. **Führen Sie das Skript aus:**
110     - Öffnen Sie ein Terminal oder eine Eingabeaufforderung.
111     - Navigieren Sie zu dem Verzeichnis, das das Skript enthält.
112     - Führen Sie das Skript mit folgendem Befehl aus:
113
114         ``bash
115         python berechne_prozentuale_anteile.py
116         ``
117
118 3. **Ergebnisse:**
119     - Die prozentualen Anteile der Straftaten pro Bezirk werden in der Konsole
          angezeigt.
120     - Außerdem wird eine neue Excel-Datei 'Prozentuale_Anteile_Straftaten.xlsx
          ' mit den Ergebnissen erstellt.
121
122 ### Beispielausgabe
123
124 ```
125 Prozentualer Anteil der Straftaten nach Bezirk:
126
127     Bezirke   Straftaten_insgesamt   Prozentualer_Anteil
128 0           Mitte                    83611                15.58
129 1   Friedrichshain-Kreuzberg          55838                10.38
130 2           Pankow                    39164                 7.29
131 3   Charlottenburg-Wilmersdorf         53630                 9.99
132 4           Spandau                    25564                 4.75
133 5   Steglitz-Zehlendorf                23719                 4.40
134 6   Tempelhof-Schöneberg              41776                 7.78
135 7           Neukölln                   45313                 8.45
136 8   Treptow-Köpenick                  26099                 4.86
137 9   Marzahn-Hellersdorf                26649                 4.96

```

```

137 10          Lichtenberg          31563          5.88
138 11          Reinickendorf        27344          5.10
139
140 Die Ergebnisse wurden in 'Prozentuale_Anteile_Straftaten.xlsx' gespeichert.
141 '''
142
143 Dieses Skript bietet eine einfache und effektive Möglichkeit, die Verteilung
    der Straftaten auf die Bezirke von Berlin zu analysieren. Sie können das
    Skript nach Bedarf anpassen, um zusätzliche Analysen durchzuführen oder
    die Ausgabe zu formatieren.

```

Listing 99: AI Response Testfall 3 Prompt 1 Ausführung 4

```

1 Gerne erstelle ich dir ein Python-Skript, das den prozentualen Anteil der
    gesamten Straftaten der einzelnen Bezirke im Vergleich zu den gesamten
    Straftaten von ganz Berlin berechnet. Dazu verwenden wir die Bibliothek '
    pandas', die sich hervorragend für die Datenanalyse eignet.
2
3 ### Voraussetzungen
4 - Stelle sicher, dass du Python installiert hast. Falls nicht, kannst du es [
    hier](https://www.python.org/downloads/) herunterladen und installieren.
5 - Installiere die 'pandas'-Bibliothek, falls noch nicht vorhanden, mit dem
    folgenden Befehl:
6
7 ```bash
8 pip install pandas
9 ```
10
11 ### Python-Skript
12
13 Hier ist das vollständige Skript mit detaillierten Kommentaren auf Deutsch:
14
15 ```python
16 import pandas as pd
17
18 def berechne_prozentualer_anteil_straftaten():
19     try:
20         # Excel-Datei laden
21         dateipfad = 'Fallzahlen.xlsx' # Pfad zur Excel-Datei
22         sheet_name = 'Fallzahlen_2023' # Name des Sheets
23
24         df = pd.read_excel(dateipfad, sheet_name=sheet_name)
25
26         # Überprüfen, ob die benötigten Spalten vorhanden sind
27         erforderliche_spalten = ['Bezirke', 'Straftaten_insgesamt']
28         fehlende_spalten = [spalte for spalte in erforderliche_spalten if
29                             spalte not in df.columns]
30         if fehlende_spalten:
31             raise ValueError(f"Die folgenden benötigten Spalten fehlen im
32                               Excel-Sheet: {fehlende_spalten}")
33
34         # Gesamte Straftaten aus der Gesamtzeile holen
35         gesamt_row = df[df['Bezirke'] == 'Berlin (PKS gesamt)']

```

```

34         if not gesamt_row.empty:
35             gesamt_straftaten = gesamt_row['Straftaten_insgesamt'].iloc[0]
36             print(f"Gesamte Straftaten in Berlin: {gesamt_straftaten}")
37         else:
38             # Falls die Gesamtzeile nicht vorhanden ist, summiere alle
Bezirkswerte
39             gesamt_straftaten = df[~df['Bezirke'].isin(['Stadtgebiet Berlin,
nicht zuzuordnen'])]['Straftaten_insgesamt'].sum()
40             print(f"Gesamte Straftaten in Berlin (berechnet): {
gesamt_straftaten}")
41
42             # Bezirke ausschließen, die nicht zugeordnet werden können oder die
Gesamtzeile sind
43             bezirke_df = df[~df['Bezirke'].isin(['Berlin (PKS gesamt)', '
Stadtgebiet Berlin, nicht zuzuordnen'])]
44
45             # Prozentualen Anteil berechnen
46             bezirke_df = bezirke_df.copy() # Kopie erstellen, um
SettingWithCopyWarning zu vermeiden
47             bezirke_df['Prozentualer_Anteil (%)'] = (bezirke_df['
Straftaten_insgesamt'] / gesamt_straftaten) * 100
48
49             # Prozentualen Anteil auf 2 Dezimalstellen runden
50             bezirke_df['Prozentualer_Anteil (%)'] = bezirke_df['
Prozentualer_Anteil (%)'].round(2)
51
52             # Ergebnis anzeigen
53             print("\nProzentualer Anteil der Straftaten pro Bezirk:")
54             print(bezirke_df[['Bezirke', 'Straftaten_insgesamt', '
Prozentualer_Anteil (%)']])
55
56             # Optional: Ergebnis in eine neue Excel-Datei speichern
57             ausgabe_datei = 'Straftaten_Prozentual.xlsx'
58             bezirke_df[['Bezirke', 'Straftaten_insgesamt', 'Prozentualer_Anteil
(%)']].to_excel(ausgabe_datei, index=False)
59             print(f"\nDas Ergebnis wurde in '{ausgabe_datei}' gespeichert.")
60
61         except FileNotFoundError:
62             print("Fehler: Die Datei 'Fallzahlen.xlsx' wurde nicht gefunden.
Bitte überprüfe den Dateipfad.")
63         except ValueError as ve:
64             print(f"Fehler: {ve}")
65         except Exception as e:
66             print(f"Ein unerwarteter Fehler ist aufgetreten: {e}")
67
68     if __name__ == "__main__":
69         berechne_prozentualer_anteil_straftaten()
70     '''
71
72     ### Erklärung des Skripts
73
74     1. **Importieren der Bibliothek:**

```

```

75 - 'pandas' wird importiert, um die Excel-Daten zu verarbeiten.
76
77 2. **Funktion 'berechne_prozentualer_anteil_straftaten':**
78 - **Daten einlesen:** Die Excel-Datei 'Fallzahlen.xlsx' wird im Sheet '
Fallzahlen_2023' eingelesen.
79 - **Spaltenüberprüfung:** Es wird überprüft, ob die benötigten Spalten ('
Bezirke' und 'Straftaten_insgesamt') vorhanden sind.
80 - **Gesamtsumme ermitteln:** Die Gesamtsumme der Straftaten wird entweder
aus der Zeile 'Berlin (PKS gesamt)' entnommen oder durch Summe der
Bezirkswerte berechnet, falls die Gesamtzeile nicht vorhanden ist.
81 - **Bezirke filtern:** Bezirke wie 'Berlin (PKS gesamt)' und 'Stadtgebiet
Berlin, nicht zuzuordnen' werden aus der Berechnung ausgeschlossen.
82 - **Prozentualen Anteil berechnen:** Für jeden Bezirk wird der prozentuale
Anteil der Straftaten berechnet und auf zwei Dezimalstellen gerundet.
83 - **Ergebnis anzeigen und speichern:** Das Ergebnis wird sowohl im
Terminal angezeigt als auch optional in eine neue Excel-Datei '
Straftaten_Prozentual.xlsx' gespeichert.
84
85 3. **Fehlerbehandlung:**
86 - Das Skript enthält Fehlerbehandlungen für fehlende Dateien, fehlende
Spalten und unerwartete Fehler, die hilfreiche Fehlermeldungen ausgeben.
87
88 ### Ausführen des Skripts
89
90 1. **Speichere das Skript:**
91 - Speichere das obige Skript in einer Datei, zum Beispiel '
berechne_straftaten.py'.
92
93 2. **Führe das Skript aus:**
94 - Öffne das Terminal oder die Eingabeaufforderung.
95 - Navigiere zum Verzeichnis, in dem sich das Skript und die Excel-Datei
befinden.
96 - Führe das Skript mit dem folgenden Befehl aus:
97
98 ```bash
99 python berechne_straftaten.py
100 ```
101
102 3. **Ergebnis:**
103 - Das Skript zeigt den prozentualen Anteil der Straftaten pro Bezirk im
Terminal an.
104 - Zusätzlich wird eine neue Excel-Datei 'Straftaten_Prozentual.xlsx'
erstellt, die diese Informationen enthält.
105
106 ### Beispielausgabe
107
108 ```
109 Gesamte Straftaten in Berlin: 536695
110
111 Prozentualer Anteil der Straftaten pro Bezirk:
112
113     Bezirke    Straftaten_insgesamt    Prozentualer_Anteil
114
115     (%)

```

```

113 0               Mitte               83611               15.57
114 1      Friedrichshain-Kreuzberg      55838
      10.40
115 2               Pankow               39164               7.30
116 3      Charlottenburg-Wilmersdorf      53630
      10.00
117 4               Spandau               25564               4.77
118 5      Steglitz-Zehlendorf               23719               4.42
119 6      Tempelhof-Schöneberg               41776               7.78
120 7               Neukölln               45313               8.46
121 8      Treptow-Köpenick               26099               4.86
122 9      Marzahn-Hellersdorf               26649               4.97
123 10              Lichtenberg              31563               5.89
124 11              Reinickendorf              27344               5.10
125
126 Das Ergebnis wurde in 'Straftaten_Prozentual.xlsx' gespeichert.
127 '''
128
129 ### Hinweise
130
131 - **Dateipfade:** Stelle sicher, dass sich die Datei 'Fallzahlen.xlsx' im
      gleichen Verzeichnis wie das Skript befindet oder gib den vollständigen
      Pfad zur Datei an.
132 - **Excel-Format:** Das Skript geht davon aus, dass die Excel-Datei das
      richtige Format hat und die genannten Spalten enthält.
133 - **Erweiterungen:** Du kannst das Skript anpassen, um weitere Analysen
      durchzuführen oder die Ausgabe an deine Bedürfnisse anzupassen.
134
135 Ich hoffe, dieses Skript hilft dir weiter! Wenn du Fragen hast oder weitere
      Anpassungen benötigst, stehe ich gerne zur Verfügung.

```

Listing 100: AI Response Testfall 3 Prompt 1 Ausführung 5

```

1 Natürlich! Hier ist ein Python-Skript, das die gewünschten prozentualen
      Anteile der 'Straftaten_insgesamt' für jeden Bezirk in Berlin berechnet.
      Das Skript verwendet die 'pandas'-Bibliothek, um die Excel-Datei zu
      verarbeiten.
2
3 ### Voraussetzungen
4
5 Stellen Sie sicher, dass Sie die 'pandas'- und 'openpyxl'-Bibliotheken
      installiert haben. Sie können sie mit 'pip' installieren, falls noch nicht
      geschehen:
6
7 '''bash
8 pip install pandas openpyxl
9 '''
10
11 ### Python-Skript
12
13 '''python
14 import pandas as pd
15

```



```

16 # Pfad zur Excel-Datei
17 excel_datei = 'Fallzahlen.xlsx'
18
19 # Name des Sheets
20 sheet_name = 'Fallzahlen_2023'
21
22 # Lesen der Excel-Datei
23 df = pd.read_excel(excel_datei, sheet_name=sheet_name)
24
25 # Überprüfen, ob die notwendigen Spalten vorhanden sind
26 notwendige_spalten = ['Bezirke', 'Straftaten_insgesamt']
27 for spalte in notwendige_spalten:
28     if spalte not in df.columns:
29         raise ValueError(f"Die Spalte '{spalte}' fehlt in der Excel-Datei.")
30
31 # Extrahieren der Gesamtanzahl von ganz Berlin
32 gesamt_row = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
33
34 if gesamt_row.empty:
35     raise ValueError("Die Gesamtzeile 'Berlin (PKS gesamt)' wurde nicht
36     gefunden.")
37
38 gesamt_straftaten = gesamt_row['Straftaten_insgesamt'].values[0]
39
40 # Entfernen der Gesamtzeile aus dem DataFrame
41 bezirk_df = df[df['Bezirke'] != 'Berlin (PKS gesamt)'].copy()
42
43 # Berechnung des prozentualen Anteils
44 bezirk_df['Prozent_Straftaten_insgesamt'] = (bezirk_df['Straftaten_insgesamt']
45     ] / gesamt_straftaten) * 100
46
47 # Optional: Runde die Prozentwerte auf zwei Dezimalstellen
48 bezirk_df['Prozent_Straftaten_insgesamt'] = bezirk_df['
49     Prozent_Straftaten_insgesamt'].round(2)
50
51 # Ausgabe des Ergebnisses
52 print(bezirk_df[['Bezirke', 'Straftaten_insgesamt', '
53     Prozent_Straftaten_insgesamt']])
54
55 # Optional: Speichern in einer neuen Excel-Datei
56 ausgabe_datei = 'Fallzahlen_mit_Prozentsatz.xlsx'
57 bezirk_df.to_excel(ausgabe_datei, index=False)
58 print(f"\nDie Ergebnisse wurden in '{ausgabe_datei}' gespeichert.")
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99

```

### Erklärung des Skripts

1. **Bibliotheken importieren:**
  - 'pandas' wird verwendet, um die Excel-Datei zu lesen und zu verarbeiten.
2. **Excel-Datei lesen:**
  - Das Skript liest die Excel-Datei 'Fallzahlen.xlsx' und das spezifische

```

Sheet 'Fallzahlen_2023'.
64
65 3. **Validierung:**
66 - Es überprüft, ob die erforderlichen Spalten ('Bezirke' und '
Straftaten_insgesamt') vorhanden sind.
67 - Es stellt sicher, dass die Gesamtzeile 'Berlin (PKS gesamt)' existiert
.
68
69 4. **Berechnung der Prozentwerte:**
70 - Die Gesamtanzahl der Straftaten in ganz Berlin ('gesamt_straftaten')
wird extrahiert.
71 - Die Gesamtzeile wird aus dem DataFrame entfernt, sodass nur die
einzelnen Bezirke übrig bleiben.
72 - Für jeden Bezirk wird der prozentuale Anteil der 'Straftaten_insgesamt'
berechnet.
73
74 5. **Ausgabe und Speicherung:**
75 - Die Ergebnisse werden im Terminal angezeigt.
76 - Optional können die Ergebnisse in eine neue Excel-Datei '
Fallzahlen_mit_Prozentsatz.xlsx' gespeichert werden.
77
78 ### Beispielausgabe
79
80 Nach dem Ausführen des Skripts erhalten Sie eine Ausgabe ähnlich der
folgenden:
81
82 '''
83
84      Bezirke  Straftaten_insgesamt
85      Prozent_Straftaten_insgesamt
86 0           Mitte                83611
87      15.58
88 1  Friedrichshain-Kreuzberg        55838
89      10.40
90 2           Pankow                39164
91      9.99
92 3  Charlottenburg-Wilmersdorf      53630
93      9.99
94 4           Spandau                25564
95      4.77
96 5  Steglitz-Zehlendorf             23719
97      4.43
98 6  Tempelhof-Schöneberg            41776
99      7.79
100 7           Neukölln             45313
101      8.47
102 8  Treptow-Köpenick               26099
103      4.87
104 9  Marzahn-Hellersdorf            26649
105      4.97
106 10          Lichtenberg           31563
107      5.89
108 11         Reinickendorf           27344
109      5.10
110 12 Stadtgebiet Berlin, nicht zuzuordnen 56425
111      10.52
112
113 Die Ergebnisse wurden in 'Fallzahlen_mit_Prozentsatz.xlsx' gespeichert.

```

```

99 '''
100
101 ### Hinweise
102
103 - Stellen Sie sicher, dass die Excel-Datei 'Fallzahlen.xlsx' im gleichen
    Verzeichnis wie das Skript liegt oder geben Sie den vollständigen Pfad zur
    Datei an.
104 - Das Skript speichert die Ergebnisse in einer neuen Excel-Datei namens '
    Fallzahlen_mit_Prozentsatz.xlsx'. Sie können den Dateinamen nach Bedarf ä
    ndern.
105 - Die Prozentwerte sind auf zwei Dezimalstellen gerundet. Sie können das
    Rundungsverhalten anpassen, indem Sie die '.round(2)' Methode ändern oder
    entfernen.
106
107 Falls Sie weitere Anpassungen oder Erweiterungen benötigen, lassen Sie es
    mich wissen!

```

Listing 101: AI Response Testfall 3 Prompt 2 Ausführung 1

```

1 Natürlich! Hier ist ein Python-Skript, das die gewünschten Berechnungen
    durchführt. Das Skript verwendet die Bibliothek 'pandas', um die Excel-
    Datei zu verarbeiten. Stellen Sie sicher, dass Sie 'pandas' und 'openpyxl'
    installiert haben. Falls nicht, können Sie sie mit 'pip' installieren.
2
3 ### Schritt 1: Installation der notwendigen Bibliotheken
4
5 Falls Sie 'pandas' und 'openpyxl' noch nicht installiert haben, können Sie
    dies über die Kommandozeile tun:
6
7 '''bash
8 pip install pandas openpyxl
9 '''
10
11 ### Schritt 2: Python-Skript zur Berechnung der prozentualen Anteile
12
13 '''python
14 import pandas as pd
15
16 # Pfad zur Excel-Datei
17 excel_datei = 'Fallzahlen.xlsx'
18
19 # Name des Sheets
20 sheet_name = 'Fallzahlen_2023'
21
22 # Laden der Excel-Datei
23 try:
24     df = pd.read_excel(excel_datei, sheet_name=sheet_name)
25 except FileNotFoundError:
26     print(f"Die Datei '{excel_datei}' wurde nicht gefunden.")
27     exit(1)
28 except Exception as e:
29     print(f"Ein Fehler ist aufgetreten: {e}")
30     exit(1)

```

```

31
32 # Sicherstellen, dass die notwendigen Spalten vorhanden sind
33 erforderliche_spalten = ['Bezirke', 'Straftaten_insgesamt']
34 for spalte in erforderliche_spalten:
35     if spalte not in df.columns:
36         print(f"Die Spalte '{spalte}' fehlt in der Excel-Datei.")
37         exit(1)
38
39 # Extrahieren der Gesamtanzahl von ganz Berlin
40 gesamt_zeile = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
41
42 if gesamt_zeile.empty:
43     print("Die Zeile 'Berlin (PKS gesamt)' wurde nicht gefunden.")
44     exit(1)
45
46 # Annahme: Es gibt genau eine Zeile mit 'Berlin (PKS gesamt)'
47 gesamt_straftaten = gesamt_zeile['Straftaten_insgesamt'].values[0]
48
49 if gesamt_straftaten == 0:
50     print("Die Gesamtanzahl der Straftaten für Berlin ist 0. Division durch Null ist nicht möglich.")
51     exit(1)
52
53 # Filtern der Bezirke (ohne die Gesamtzeile)
54 bezirke_df = df[df['Bezirke'] != 'Berlin (PKS gesamt)'].copy()
55
56 # Berechnung des prozentualen Anteils
57 bezirke_df['Prozentualer_Anteil (%)'] = (bezirke_df['Straftaten_insgesamt'] /
58     gesamt_straftaten) * 100
59
60 # Optional: Runden auf zwei Dezimalstellen
61 bezirke_df['Prozentualer_Anteil (%)'] = bezirke_df['Prozentualer_Anteil (%)']
62     .round(2)
63
64 # Sortieren nach dem prozentualen Anteil absteigend
65 bezirke_df.sort_values(by='Prozentualer_Anteil (%)', ascending=False, inplace
66     =True)
67
68 # Ergebnis anzeigen
69 print(bezirke_df[['Bezirke', 'Straftaten_insgesamt', 'Prozentualer_Anteil (%)
70     ']])
71
72 # Optional: Speichern in eine neue Excel-Datei
73 output_datei = 'Fallzahlen_prozentual.xlsx'
74 try:
75     bezirke_df.to_excel(output_datei, sheet_name='Prozentualer_Anteil', index
76         =False)
77     print(f"\nDie Ergebnisse wurden in '{output_datei}' gespeichert.")
78 except Exception as e:
79     print(f"Fehler beim Speichern der Datei: {e}")
80
81 '''

```

```

77 ### Erklärung des Skripts
78
79 1. **Importieren der Bibliothek:**
80 - 'pandas' wird verwendet, um die Excel-Datei zu lesen und die Daten zu
    bearbeiten.
81
82 2. **Laden der Excel-Datei:**
83 - Das Skript lädt die Datei 'Fallzahlen.xlsx' und das spezifische Sheet '
    Fallzahlen_2023'.
84 - Es gibt Fehlermeldungen aus, wenn die Datei nicht gefunden wird oder ein
    anderer Fehler auftritt.
85
86 3. **Überprüfen der erforderlichen Spalten:**
87 - Das Skript stellt sicher, dass die Spalten 'Bezirke' und '
    Straftaten_insgesamt' vorhanden sind.
88
89 4. **Extrahieren der Gesamtanzahl von Straftaten:**
90 - Es sucht die Zeile, in der der Bezirk 'Berlin (PKS gesamt)' ist, und
    extrahiert den Wert der Spalte 'Straftaten_insgesamt'.
91
92 5. **Berechnung des prozentualen Anteils:**
93 - Für jeden Bezirk wird der Anteil der Straftaten berechnet im Verhältnis
    zur Gesamtanzahl von Berlin.
94 - Das Ergebnis wird auf zwei Dezimalstellen gerundet.
95
96 6. **Sortieren und Anzeigen der Ergebnisse:**
97 - Die Bezirke werden nach ihrem prozentualen Anteil sortiert, beginnend
    mit dem höchsten Anteil.
98 - Das Skript gibt die Bezirke, die Straftaten insgesamt und den
    prozentualen Anteil aus.
99
100 7. **Speichern der Ergebnisse:**
101 - Optional speichert das Skript die Ergebnisse in einer neuen Excel-Datei
    namens 'Fallzahlen_prozentual.xlsx' im Sheet 'Prozentualer_Anteil'.
102
103 ### Beispielausgabe
104
105 Nach dem Ausführen des Skripts könnte die Ausgabe wie folgt aussehen:
106
107 '''
108
109
110
111
112
113
114
115
116

```

	Bezirke	Straftaten_insgesamt	Prozentualer_Anteil
(%)			
0	Mitte	83611	15.58
1	Friedrichshain-Kreuzberg	55838	10.40
2	Pankow	39164	7.31
3	Charlottenburg-Wilmersdorf	53630	10.00
4	Spandau	25564	4.76
5	Steglitz-Zehlendorf	23719	4.42
6	Tempelhof-Schöneberg	41776	7.78
7	Neukölln	45313	8.47

```

117 8          Treptow-Köpenick          26099          4.87
118 9          Marzahn-Hellersdorf        26649          4.97
119 10         Lichtenberg                31563          5.89
120 11         Reinickendorf              27344          5.10
121 12 Stadtgebiet Berlin, nicht zuzuordnen 56425
    10.54
122
123 Die Ergebnisse wurden in 'Fallzahlen_prozentual.xlsx' gespeichert.
124 '''
125
126 ### Hinweise
127
128 - **Anpassung der Pfade:** Stellen Sie sicher, dass sich das Python-Skript im
    gleichen Verzeichnis wie die Excel-Datei befindet oder passen Sie den
    Pfad zur Excel-Datei entsprechend an.
129 - **Weitere Berechnungen:** Wenn Sie zusätzliche Berechnungen durchführen mö-
    chten (z.B. für andere Spalten wie 'Raub', 'Straßenraub' etc.), können Sie
    ähnliche Berechnungen hinzufügen.
130 - **Visualisierung:** Für eine bessere Übersicht können Sie die Ergebnisse
    auch grafisch darstellen, z.B. mit einem Balkendiagramm. Dafür können
    Bibliotheken wie 'matplotlib' oder 'seaborn' verwendet werden.
131
132 Falls Sie weitere Anpassungen oder zusätzliche Funktionen benötigen, lassen
    Sie es mich wissen!

```

Listing 102: AI Response Testfall 3 Prompt 2 Ausführung 2

```

1 Hier ist ein Python-Skript, das den prozentualen Anteil der '
    Straftaten_insgesamt' für jeden Bezirk in Berlin berechnet. Das Skript
    verwendet die Bibliothek 'pandas', um die Excel-Datei zu lesen und die
    Berechnungen durchzuführen.
2
3 **Schritte des Skripts:**
4
5 1. **Import der notwendigen Bibliotheken:** Wir verwenden 'pandas' für die
    Datenverarbeitung.
6 2. **Lesen der Excel-Datei:** Das Skript liest das Sheet 'Fallzahlen_2023'
    aus der Datei 'Fallzahlen.xlsx'.
7 3. **Identifizieren des Gesamtwerts:** Der Gesamtwert für Berlin wird aus der
    Zeile mit 'Bezirke' als 'Berlin (PKS gesamt)' entnommen.
8 4. **Berechnen der prozentualen Anteile:** Für jeden Bezirk wird der Anteil
    der 'Straftaten_insgesamt' am Gesamtwert berechnet.
9 5. **Speichern der Ergebnisse:** Die Ergebnisse werden als neue Spalte
    hinzugefügt und in einer neuen Excel-Datei gespeichert.
10
11 **Voraussetzungen:**
12
13 - Stelle sicher, dass die Bibliothek 'pandas' installiert ist. Falls nicht,
    kannst du sie mit 'pip install pandas' installieren.
14 - Die Excel-Datei 'Fallzahlen.xlsx' sollte sich im selben Verzeichnis wie das
    Skript befinden oder du musst den vollständigen Pfad zur Datei angeben.
15
16 **Python-Skript:**

```

```

17
18 '''python
19 import pandas as pd
20
21 # Pfad zur Excel-Datei
22 excel_datei = 'Fallzahlen.xlsx'
23
24 # Name des Sheets
25 sheet_name = 'Fallzahlen_2023'
26
27 # Lesen des Excel-Sheets
28 try:
29     df = pd.read_excel(excel_datei, sheet_name=sheet_name)
30 except FileNotFoundError:
31     print(f"Die Datei '{excel_datei}' wurde nicht gefunden.")
32     exit(1)
33 except Exception as e:
34     print(f"Ein Fehler ist aufgetreten beim Lesen der Excel-Datei: {e}")
35     exit(1)
36
37 # Überprüfen, ob die notwendigen Spalten vorhanden sind
38 if 'Bezirke' not in df.columns or 'Straftaten_insgesamt' not in df.columns:
39     print("Die erforderlichen Spalten 'Bezirke' oder 'Straftaten_insgesamt'
40         fehlen in der Tabelle.")
41     exit(1)
42
43 # Finden des Gesamtwerts für Berlin
44 gesamt_zeile = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
45
46 if gesamt_zeile.empty:
47     print("Die Gesamtzeile 'Berlin (PKS gesamt)' wurde nicht gefunden.")
48     exit(1)
49
50 gesamt_straftaten = gesamt_zeile['Straftaten_insgesamt'].values[0]
51
52 # Entfernen der Gesamtzeile aus den Bezirken
53 bezirke_df = df[df['Bezirke'] != 'Berlin (PKS gesamt)'].copy()
54
55 # Berechnen des prozentualen Anteils
56 bezirke_df['Prozentualer_Anteil'] = (bezirke_df['Straftaten_insgesamt'] /
57     gesamt_straftaten) * 100
58
59 # Optional: Runden auf zwei Dezimalstellen
60 bezirke_df['Prozentualer_Anteil'] = bezirke_df['Prozentualer_Anteil'].round
61     (2)
62
63 # Anzeigen der Ergebnisse
64 print(bezirke_df[['Bezirke', 'Straftaten_insgesamt', 'Prozentualer_Anteil']])
65
66 # Speichern der Ergebnisse in einer neuen Excel-Datei
67 ausgabe_datei = 'Fallzahlen_Prozente.xlsx'
68 try:

```

```

66     bezirke_df.to_excel(ausgabe_datei, sheet_name='Prozentuale_Fallzahlen',
67                        index=False)
68     print(f"\nDie prozentualen Anteile wurden in '{ausgabe_datei}'
69           gespeichert.")
70 except Exception as e:
71     print(f"Ein Fehler ist aufgetreten beim Speichern der Excel-Datei: {e}")
72
73
74 **Erklärung der wichtigsten Teile des Skripts:**
75
76 - **Lesen der Excel-Datei:**
77     '''python
78     df = pd.read_excel(excel_datei, sheet_name=sheet_name)
79     '''
80     Dies liest das angegebene Sheet der Excel-Datei in ein DataFrame.
81
82 - **Identifizieren des Gesamtwerts:**
83     '''python
84     gesamt_zeile = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
85     gesamt_straftaten = gesamt_zeile['Straftaten_insgesamt'].values[0]
86     '''
87     Hier wird die Zeile gefunden, die den Gesamtwert für Berlin enthält.
88
89 - **Berechnen des prozentualen Anteils:**
90     '''python
91     bezirke_df['Prozentualer_Anteil'] = (bezirke_df['Straftaten_insgesamt'] /
92                                         gesamt_straftaten) * 100
93     bezirke_df['Prozentualer_Anteil'] = bezirke_df['Prozentualer_Anteil'].round
94     (2)
95     '''
96     Diese Zeilen berechnen den prozentualen Anteil und runden ihn auf zwei
97     Dezimalstellen.
98
99 - **Speichern der Ergebnisse:**
100     '''python
101     bezirke_df.to_excel(ausgabe_datei, sheet_name='Prozentuale_Fallzahlen',
102                        index=False)
103     '''
104     Die Ergebnisse werden in eine neue Excel-Datei mit einem neuen Sheet
105     gespeichert.
106
107 **Ausgabe:**
108
109 Das Skript gibt die Bezirke zusammen mit den 'Straftaten_insgesamt' und ihrem
110 prozentualen Anteil aus und speichert die Ergebnisse in der Datei '
111 Fallzahlen_Prozente.xlsx'.
112
113
114 **Beispielausgabe:**
115     '''
116
117             Bezirke    Straftaten_insgesamt    Prozentualer_Anteil
118 0             Mitte              83611              15.58
119 1    Friedrichshain-Kreuzberg              55838              10.41

```



109	2	Pankow	39164	7.31
110	3	Charlottenburg-Wilmersdorf	53630	10.00
111	4	Spandau	25564	4.77
112	5	Steglitz-Zehlendorf	23719	4.42
113	6	Tempelhof-Schöneberg	41776	7.80
114	7	Neukölln	45313	8.47
115	8	Treptow-Köpenick	26099	4.87
116	9	Marzahn-Hellersdorf	26649	4.97
117	10	Lichtenberg	31563	5.89
118	11	Reinickendorf	27344	5.10
119	12	Stadtgebiet Berlin, nicht zuzuordnen	56425	
120		10.53		

121 Die prozentualen Anteile wurden in 'Fallzahlen\_Prozente.xlsx' gespeichert.  
 122 '''  
 123  
 124 Dieses Skript bietet eine einfache Möglichkeit, die Verteilung der Straftaten über die Bezirke Berlins hinweg zu analysieren. Du kannst es nach Bedarf weiter anpassen, z.B. zusätzliche Analysen hinzufügen oder die Ergebnisse in einem anderen Format speichern.

Listing 103: AI Response Testfall 3 Prompt 2 Ausführung 3

```

1 Hier ist ein Python-Skript, das den prozentualen Anteil der **'
  Straftaten_insgesamt'** für jeden Bezirk von ganz Berlin berechnet. Das
  Skript liest die Excel-Datei '**Fallzahlen.xlsx**', verarbeitet die Daten
  im Sheet '**Fallzahlen_2023**' und speichert die Ergebnisse in einer neuen
  Excel-Datei.

2
3 ## Voraussetzungen
4
5 Stelle sicher, dass du Python installiert hast. Außerdem benötigst du die
  Bibliotheken '**pandas**' und '**openpyxl**'. Du kannst sie mit folgendem
  Befehl installieren:

6
7 '''bash
8 pip install pandas openpyxl
9 '''

10
11 ## Python-Skript
12
13 '''python
14 import pandas as pd

15
16 # Pfad zur Excel-Datei und Name des Sheets
17 excel_datei = 'Fallzahlen.xlsx'
18 blatt_name = 'Fallzahlen_2023'
19
20 # Lade die Daten aus der Excel-Datei
21 df = pd.read_excel(excel_datei, sheet_name=blatt_name)
22
23 # Finde die Gesamtzahl der Straftaten für ganz Berlin
24 berlin_gesamtreihe = df[df['Bezirke'] == 'Berlin (PKS gesamt)']

```

```

25 if berlin_gesamtreihe.empty:
26     raise ValueError("Die Zeile 'Berlin (PKS gesamt)' wurde in den Daten
    nicht gefunden.")
27
28 berlin_gesamt = berlin_gesamtreihe['Straftaten_insgesamt'].iloc[0]
29
30 # Entferne die Gesamtzeile aus dem DataFrame
31 bezirke_df = df[df['Bezirke'] != 'Berlin (PKS gesamt)'].copy()
32
33 # Berechne den prozentualen Anteil der Straftaten für jeden Bezirk
34 bezirke_df['Prozentualer_Anteil'] = ((bezirke_df['Straftaten_insgesamt'] /
    berlin_gesamt) * 100).round(2)
35
36 # Ausgabe der Ergebnisse
37 print(bezirke_df[['Bezirke', 'Straftaten_insgesamt', 'Prozentualer_Anteil']])
38
39 # Speichere das Ergebnis in eine neue Excel-Datei
40 bezirke_df.to_excel('Fallzahlen_mit_Prozentanteilen.xlsx', index=False)
41 '''
42
43 ## Erklärung des Skripts
44
45 1. **Import der Bibliothek:**
46     '''python
47     import pandas as pd
48     '''
49
50 2. **Definition von Datei- und Blattnamen:**
51     '''python
52     excel_datei = 'Fallzahlen.xlsx'
53     blatt_name = 'Fallzahlen_2023'
54     '''
55
56 3. **Laden der Excel-Daten:**
57     '''python
58     df = pd.read_excel(excel_datei, sheet_name=blatt_name)
59     '''
60     Dies liest die Daten aus dem angegebenen Sheet in ein Pandas DataFrame ein
61     .
62
63 4. **Ermittlung der Gesamtanzahl der Straftaten für ganz Berlin:**
64     '''python
65     berlin_gesamtreihe = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
66     if berlin_gesamtreihe.empty:
67         raise ValueError("Die Zeile 'Berlin (PKS gesamt)' wurde in den Daten
            nicht gefunden.")
68
69     berlin_gesamt = berlin_gesamtreihe['Straftaten_insgesamt'].iloc[0]
70     '''
71     Hier wird nach der Zeile gesucht, die die Gesamtanzahl der Straftaten für
    Berlin enthält. Falls diese Zeile nicht gefunden wird, gibt das Skript
    einen Fehler aus.

```

```

71
72 5. **Ausschluss der Gesamtzeile aus den Bezirksdaten:**
73   '''python
74   bezirke_df = df[df['Bezirke'] != 'Berlin (PKS gesamt)'].copy()
75   '''
76   Dies filtert die Zeile mit den Gesamtdaten heraus, sodass nur die
       einzelnen Bezirke übrig bleiben.
77
78 6. **Berechnung des prozentualen Anteils:**
79   '''python
80   bezirke_df['Prozentualer_Anteil'] = ((bezirke_df['Straftaten_insgesamt'] /
       berlin_gesamt) * 100).round(2)
81   '''
82   Für jeden Bezirk wird der Anteil der Straftaten an der Gesamtanzahl
       berechnet und auf zwei Dezimalstellen gerundet.
83
84 7. **Ausgabe der Ergebnisse:**
85   '''python
86   print(bezirke_df[['Bezirke', 'Straftaten_insgesamt', 'Prozentualer_Anteil'
       ]])
87   '''
88   Dies zeigt eine Tabelle mit den Bezirken, der Anzahl der Straftaten und
       dem prozentualen Anteil an.
89
90 8. **Speichern der Ergebnisse in einer neuen Excel-Datei:**
91   '''python
92   bezirke_df.to_excel('Fallzahlen_mit_Prozentanteilen.xlsx', index=False)
93   '''
94   Die berechneten Daten werden in einer neuen Excel-Datei gespeichert.
95
96 ## Ausführung des Skripts
97
98 1. **Speichere das Skript:**
99   Speichere den obigen Code in einer Datei, zum Beispiel '''
       berechne_prozente.py'''.
100
101 2. **Führe das Skript aus:**
102   Öffne ein Terminal oder eine Eingabeaufforderung und navigiere zum
       Verzeichnis, in dem sich das Skript befindet. Führe dann folgenden Befehl
       aus:
103
104   '''bash
105   python berechne_prozente.py
106   '''
107
108 3. **Ergebnis:**
109   Das Skript gibt die Bezirke mit deren '''Straftaten_insgesamt''' und dem
       '''Prozentualer_Anteil''' aus und erstellt eine neue Excel-Datei '''
       Fallzahlen_mit_Prozentanteilen.xlsx''' mit den berechneten Daten.
110
111 ## Beispielausgabe
112

```

```

113 Nach der Ausführung des Skripts könnte die Ausgabe ungefähr wie folgt
    aussehen:
114
115 '''
116             Bezirke   Straftaten_insgesamt   Prozentualer_Anteil
117 0             Mitte             83611             15.58
118 1   Friedrichshain-Kreuzberg             55838             10.41
119 2             Pankow             39164             7.30
120 3   Charlottenburg-Wilmersdorf             53630             10.00
121 4             Spandau             25564             4.77
122 5   Steglitz-Zehlendorf             23719             4.42
123 6   Tempelhof-Schöneberg             41776             7.79
124 7             Neukölln             45313             8.48
125 8   Treptow-Köpenick             26099             4.87
126 9   Marzahn-Hellersdorf             26649             4.97
127 10            Lichtenberg             31563             5.89
128 11            Reinickendorf             27344             5.10
129 12 Stadtgebiet Berlin, nicht zuzuordnen             56425
    10.54
130 '''
131
132 ## Hinweise
133
134 - **Fehlende Daten:** Stelle sicher, dass die Zeile **Berlin (PKS gesamt)**
    in deinem Excel-Sheet vorhanden ist. Andernfalls wirft das Skript einen
    Fehler.
135 - **Dateipfade:** Falls sich die Excel-Datei oder das Skript in
    unterschiedlichen Verzeichnissen befinden, musst du die Pfade entsprechend
    anpassen.
136 - **Erweiterungen:** Du kannst das Skript erweitern, um weitere Analysen
    durchzuführen oder die Ausgabe weiter zu formatieren.
137
138 Bei Fragen oder Problemen stehe ich gerne zur Verfügung!

```

Listing 104: AI Response Testfall 3 Prompt 2 Ausführung 4

```

1 Hier ist ein Python-Skript, das den prozentualen Anteil der gesamten
    Straftaten ('Straftaten_insgesamt') für jeden Bezirk in Berlin berechnet.
    Das Skript verwendet die Bibliotheken 'pandas' und 'openpyxl'. Stelle
    sicher, dass beide installiert sind. Du kannst sie mit folgendem Befehl
    installieren, falls noch nicht geschehen:
2
3 '''bash
4 pip install pandas openpyxl
5 '''
6
7 ### Python-Skript: 'prozentanteil_straftaten.py'
8
9 '''python
10 import pandas as pd
11
12 # Pfad zur Excel-Datei und Name des Sheets
13 excel_datei = 'Fallzahlen.xlsx'

```

```

14 sheet_name = 'Fallzahlen_2023'
15
16 # Excel-Datei einlesen
17 df = pd.read_excel(excel_datei, sheet_name=sheet_name)
18
19 # Gesamtanzahl der Straftaten in Berlin extrahieren
20 # Annahme: Der Eintrag für "Berlin (PKS gesamt)" ist eindeutig
21 gesamt_strafaten = df.loc[df['Bezirke'] == 'Berlin (PKS gesamt)', '
    Straftaten_insgesamt'].values[0]
22
23 # Entfernen der Gesamtzeile, um nur die Bezirke zu betrachten
24 df_bezirke = df[df['Bezirke'] != 'Berlin (PKS gesamt)'].copy()
25
26 # Berechnung des prozentualen Anteils der Straftaten für jeden Bezirk
27 df_bezirke['Prozent_Straftaten'] = (df_bezirke['Straftaten_insgesamt'] /
    gesamt_strafaten) * 100
28
29 # Optional: Sortieren nach dem prozentualen Anteil
30 df_bezirke.sort_values(by='Prozent_Straftaten', ascending=False, inplace=True
    )
31
32 # Ergebnis anzeigen
33 print(df_bezirke[['Bezirke', 'Straftaten_insgesamt', 'Prozent_Straftaten']])
34
35 # Optional: Ergebnisse in eine neue Excel-Datei speichern
36 ausgabe_datei = 'Fallzahlen_mit_Prozent.xlsx'
37 df_bezirke.to_excel(ausgabe_datei, index=False)
38 print(f"\nDie Ergebnisse wurden in '{ausgabe_datei}' gespeichert.")
39 '''
40
41 ### Erläuterungen zum Skript
42
43 1. **Importieren der Bibliotheken:**
44     - 'pandas' wird für die Datenverarbeitung verwendet.
45
46 2. **Einlesen der Excel-Datei:**
47     - Das Skript liest die Excel-Datei 'Fallzahlen.xlsx' und das spezifische
48       Sheet 'Fallzahlen_2023'.
49
50 3. **Extrahieren der Gesamtanzahl der Straftaten:**
51     - Es sucht die Zeile, in der der Bezirk 'Berlin (PKS gesamt)' ist, und
52       extrahiert den Wert aus der Spalte 'Straftaten_insgesamt'.
53
54 4. **Filtern der Bezirke:**
55     - Die Gesamtzeile wird entfernt, sodass nur die einzelnen Bezirke
56       verbleiben.
57
58 5. **Berechnung des prozentualen Anteils:**
59     - Für jeden Bezirk wird der prozentuale Anteil der Straftaten berechnet,
60       indem die Straftaten des Bezirks durch die Gesamtzahl der Straftaten in
61       Berlin geteilt und mit 100 multipliziert wird.

```

```

58 6. **Sortieren der Ergebnisse (optional):**
59 - Die Bezirke werden nach dem prozentualen Anteil der Straftaten
    absteigend sortiert.
60
61 7. **Ausgabe der Ergebnisse:**
62 - Das Skript druckt eine Tabelle mit den Bezirken, den gesamten Straftaten
    und dem prozentualen Anteil.
63
64 8. **Speichern der Ergebnisse (optional):**
65 - Die Ergebnisse werden in eine neue Excel-Datei 'Fallzahlen_mit_Prozent.
    xlsx' gespeichert.
66
67 ### Beispielhafte Ausgabe
68
69 '''
70
71         Bezirke  Straftaten_insgesamt  Prozent_Straftaten
72 0           Mitte                83611             15.585348
73 1   Friedrichshain-Kreuzberg        55838             10.404365
74 2           Pankow                39164             7.295930
75 3   Charlottenburg-Wilmersdorf      53630             9.980373
76 4           Spandau                25564             4.763478
77 5   Steglitz-Zehlendorf            23719             4.416960
78 6   Tempelhof-Schöneberg           41776             7.785106
79 7           Neukölln             45313             8.467841
80 8   Treptow-Köpenick               26099             4.853939
81 9   Marzahn-Hellersdorf            26649             4.976549
82 10          Lichtenberg           31563             5.889312
83 11          Reinickendorf          27344             5.112226
84 12 Stadtgebiet Berlin, nicht zuzuordnen 56425
    10.532238
85
86 Die Ergebnisse wurden in 'Fallzahlen_mit_Prozent.xlsx' gespeichert.
87 '''
88
89 ### Hinweise
90
91 - **Spaltennamen überprüfen:** Stelle sicher, dass die Spaltennamen in deinem
    Excel-Sheet exakt mit denen im Skript übereinstimmen (z.B. 'Bezirke' und
    'Straftaten_insgesamt'). Groß- und Kleinschreibung sowie Leerzeichen mü-
    ssen genau übereinstimmen.
92
93 - **Fehlerbehandlung (optional):** Für eine robustere Anwendung könntest du
    zusätzliche Fehlerprüfungen hinzufügen, z.B. um sicherzustellen, dass die
    Gesamtstraftaten-Zeile existiert oder dass keine Division durch Null
    erfolgt.
94
95 - **Anpassungen:** Du kannst das Skript nach Bedarf erweitern, z.B. durch
    Visualisierungen oder weitere Analysen.
96
97 Wenn du Fragen hast oder weitere Anpassungen benötigst, stehe ich gerne zur
    Verfügung!

```

Listing 105: AI Response Testfall 3 Prompt 2 Ausführung 5

```

1 Hier ist ein vollständiges Python-Skript, das die von Ihnen beschriebenen
  Anforderungen erfüllt. Dieses Skript verwendet die 'pandas'-Bibliothek, um
  die Excel-Datei zu verarbeiten. Stellen Sie sicher, dass Sie 'pandas' und
  'openpyxl' installiert haben, falls Sie diese noch nicht installiert
  haben. Sie können sie mit 'pip' installieren:
2
3 '''bash
4 pip install pandas openpyxl
5 '''
6
7 ### Python-Skript: 'fallzahlen_verarbeiten.py'
8
9 '''python
10 import pandas as pd
11
12 def main():
13     # 1. Excel-Datei einlesen
14     excel_datei = 'Fallzahlen.xlsx'
15     sheet_name = 'Fallzahlen_2023'
16
17     try:
18         df = pd.read_excel(excel_datei, sheet_name=sheet_name)
19         print(f"Excel-Datei '{excel_datei}' erfolgreich eingelesen.")
20     except FileNotFoundError:
21         print(f"Fehler: Die Datei '{excel_datei}' wurde nicht gefunden.")
22         return
23     except Exception as e:
24         print(f"Ein Fehler ist beim Einlesen der Excel-Datei aufgetreten: {e}")
25         return
26
27     # 2. Tabelle als Pandas DataFrame speichern
28     # (bereits durch das Einlesen mit pandas erledigt)
29
30     # 3. Überprüfen, ob die notwendigen Spalten vorhanden sind
31     erforderliche_spalten = ['Bezirke', 'Straftaten_insgesamt']
32     fehlende_spalten = [spalte for spalte in erforderliche_spalten if spalte
33                          not in df.columns]
34
35     if fehlende_spalten:
36         print(f"Fehler: Fehlende Spalten in der Excel-Datei: {
37               fehlende_spalten}")
38         return
39     else:
40         print("Alle erforderlichen Spalten sind vorhanden.")
41
42     # 4. Gesamtzahl der Straftaten für ganz Berlin finden
43     gesamt_row = df[df['Bezirke'] == 'Berlin (PKS gesamt)']

```

```

43     if gesam_row.empty:
44         print("Fehler: Die Zeile mit 'Berlin (PKS gesamt)' wurde nicht
gefunden.")
45         return
46
47     gesam_straftaten = gesam_row['Straftaten_insgesamt'].values[0]
48     print(f"Gesamtzahl der Straftaten für Berlin: {gesam_straftaten}")
49
50     # 5. Prozentualen Anteil der einzelnen Bezirke berechnen
51     # Ausschließen der Gesamtzeile für die Berechnung
52     df_bezirke = df[df['Bezirke'] != 'Berlin (PKS gesamt)'].copy()
53
54     if df_bezirke.empty:
55         print("Fehler: Es gibt keine Bezirke nach dem Ausschluss von 'Berlin
(PKS gesamt)'")
56         return
57
58     # Prozentualer Anteil berechnen
59     df_bezirke['Prozentualer_Anteil (%)'] = (df_bezirke['Straftaten_insgesamt
'] / gesam_straftaten) * 100
60
61     # Optional: Runden auf zwei Dezimalstellen
62     df_bezirke['Prozentualer_Anteil (%)'] = df_bezirke['Prozentualer_Anteil
(%)'].round(2)
63
64     print("Prozentualer Anteil der Straftaten pro Bezirk berechnet.")
65
66     # 6. Ergebnis als DataFrame zurückgeben
67     # Hier zeigen wir das Ergebnis an. Sie können es auch in eine Datei
speichern.
68     ergebnis_df = df_bezirke[['Bezirke', 'Straftaten_insgesamt', '
Prozentualer_Anteil (%)']]
69     print("\nErgebnis DataFrame:")
70     print(ergebnis_df.to_string(index=False))
71
72     # Optional: Ergebnis in eine neue Excel-Datei speichern
73     ausgabe_datei = 'Fallzahlen_prozentual.xlsx'
74     try:
75         ergebnis_df.to_excel(ausgabe_datei, index=False)
76         print(f"\nErgebnis erfolgreich in '{ausgabe_datei}' gespeichert.")
77     except Exception as e:
78         print(f"Ein Fehler ist beim Speichern der Ergebnisdatei aufgetreten:
{e}")
79
80 if __name__ == "__main__":
81     main()
82
83
84 ### Erklärung der Schritte
85
86 1. **Excel-Datei einlesen**:
87     - Die Excel-Datei 'Fallzahlen.xlsx' wird mit der Methode 'pd.read_excel'

```



```

eingeliesen.
88 - Das spezifische Sheet 'Fallzahlen_2023' wird ausgewählt.
89 - Fehlerbehandlung ist implementiert, falls die Datei nicht gefunden wird
oder ein anderes Problem auftritt.
90
91 2. **Pandas DataFrame speichern**:
92 - Das Einlesen mit 'pandas' erstellt bereits einen DataFrame namens 'df'.
93
94 3. **Überprüfen der notwendigen Spalten**:
95 - Es wird geprüft, ob die Spalten 'Bezirke' und 'Straftaten_insgesamt'
96 ' im DataFrame vorhanden sind.
- Falls Spalten fehlen, wird ein Fehler ausgegeben und das Skript beendet
.
97
98 4. **Gesamtzahl der Straftaten für Berlin finden**:
99 - Die Zeile mit dem Bezirk 'Berlin (PKS gesamt)' wird gefiltert.
100 - Die Gesamtzahl der Straftaten wird aus dieser Zeile extrahiert.
101 - Wenn diese Zeile nicht gefunden wird, wird ein Fehler ausgegeben.
102
103 5. **Prozentualen Anteil berechnen**:
104 - Die Zeile 'Berlin (PKS gesamt)' wird aus dem DataFrame ausgeschlossen
, sodass nur die einzelnen Bezirke verbleiben.
105 - Für jeden Bezirk wird der prozentuale Anteil der Straftaten an der
Gesamtzahl berechnet.
106 - Das Ergebnis wird auf zwei Dezimalstellen gerundet.
107
108 6. **Ergebnis als DataFrame zurückgeben**:
109 - Das Ergebnis-DataFrame enthält die Spalten 'Bezirke', '
Straftaten_insgesamt' und 'Prozentualer_Anteil (%)'.
110 - Das Ergebnis wird im Terminal angezeigt.
111 - Optional wird das Ergebnis in eine neue Excel-Datei '
Fallzahlen_prozentual.xlsx' gespeichert.
112
113 ### Nutzung des Skripts
114
115 1. **Speichern Sie das Skript**:
116 - Kopieren Sie das obige Skript in eine Datei namens '
fallzahlen_verarbeiten.py'.
117
118 2. **Stellen Sie sicher, dass die Excel-Datei vorhanden ist**:
119 - Platzieren Sie die Datei 'Fallzahlen.xlsx' im gleichen Verzeichnis wie
das Skript oder passen Sie den Pfad im Skript entsprechend an.
120
121 3. **Führen Sie das Skript aus**:
122 - Öffnen Sie ein Terminal oder eine Eingabeaufforderung.
123 - Navigieren Sie zum Verzeichnis, das das Skript und die Excel-Datei enth
ält.
124 - Führen Sie das Skript mit folgendem Befehl aus:
125
126 'bash
127 python fallzahlen_verarbeiten.py
128 ''

```

```

129
130 4. **Ergebnis überprüfen**:
131     - Das Skript zeigt das Ergebnis im Terminal an.
132     - Zusätzlich wird eine neue Excel-Datei 'Fallzahlen_prozentual.xlsx'
      erstellt, die das Ergebnis enthält.
133
134 ### Beispielausgabe
135
136 '''
137 Excel-Datei 'Fallzahlen.xlsx' erfolgreich eingelesen.
138 Alle erforderlichen Spalten sind vorhanden.
139 Gesamtzahl der Straftaten für Berlin: 536695
140 Prozentualer Anteil der Straftaten pro Bezirk berechnet.
141
142 Ergebnis DataFrame:
143
144      Bezirke  Straftaten_insgesamt  Prozentualer_Anteil (%)
145      Mitte                83611                15.57
146      Friedrichshain-Kreuzberg        55838                10.41
147      Pankow                39164                 7.30
148      Charlottenburg-Wilmersdorf        53630                10.00
149      Spandau                25564                 4.77
150      Steglitz-Zehlendorf            23719                 4.42
151      Tempelhof-Schöneberg            41776                 7.78
152      Neukölln                45313                 8.46
153      Treptow-Köpenick            26099                 4.87
154      Marzahn-Hellersdorf            26649                 4.97
155      Lichtenberg            31563                 5.88
156      Reinickendorf            27344                 5.10
157      Stadtgebiet Berlin, nicht zuzuordnen        56425
158      10.52
159
160 Ergebnis erfolgreich in 'Fallzahlen_prozentual.xlsx' gespeichert.
161 '''
162
163 Dieses Skript bietet eine robuste Lösung zur Verarbeitung Ihrer Excel-Daten
164 und zur Berechnung der prozentualen Anteile der Straftaten pro Bezirk in
165 Berlin. Sie können das Skript nach Bedarf erweitern oder anpassen, zum
166 Beispiel indem Sie zusätzliche Analysen hinzufügen oder das Ausgabeformat
167 ändern.

```

Listing 106: AI Response Testfall 3 Prompt 3 Ausführung 1

```

1 Hier ist ein Python-Skript, das die von Ihnen beschriebenen Anforderungen erf
  üllt. Das Skript verwendet die 'pandas'-Bibliothek, um die Excel-Datei zu
  verarbeiten.
2
3 Stellen Sie sicher, dass Sie 'pandas' und 'openpyxl' installiert haben. Sie k
  önnen diese Bibliotheken mit 'pip' installieren, falls sie noch nicht
  installiert sind:
4
5 '''bash
6 pip install pandas openpyxl
7 '''

```

```

8
9 ### Python-Skript
10
11 '''python
12 import pandas as pd
13
14 def verarbeite_fallzahlen(dateipfad: str, sheet_name: str = 'Fallzahlen_2023'
15 ) -> pd.DataFrame:
16     """
17     Liest eine Excel-Datei ein, überprüft notwendige Spalten, berechnet
18     den prozentualen Anteil der Straftaten pro Bezirk und gibt das Ergebnis
19     als DataFrame zurück.
20
21     :param dateipfad: Pfad zur Excel-Datei 'Fallzahlen.xlsx'.
22     :param sheet_name: Name des Sheets in der Excel-Datei. Standard ist '
23     Fallzahlen_2023'.
24     :return: DataFrame mit Bezirken, Straftaten insgesamt und prozentualen
25     Anteil.
26     """
27     try:
28         # 1. Excel-Datei einlesen
29         df = pd.read_excel(dateipfad, sheet_name=sheet_name)
30         print("Excel-Datei erfolgreich eingelesen.")
31     except FileNotFoundError:
32         raise FileNotFoundError(f"Die Datei '{dateipfad}' wurde nicht
33         gefunden.")
34     except Exception as e:
35         raise Exception(f"Fehler beim Einlesen der Excel-Datei: {e}")
36
37     # 2. Die Tabelle als Pandas DataFrame gespeichert (bereits in 'df')
38
39     # 3. Überprüfen, ob die notwendigen Spalten vorhanden sind
40     erforderliche_spalten = ['Bezirke', 'Straftaten_insgesamt']
41     fehlende_spalten = [spalte for spalte in erforderliche_spalten if spalte
42     not in df.columns]
43     if fehlende_spalten:
44         raise ValueError(f"Die folgenden erforderlichen Spalten fehlen in der
45         Tabelle: {fehlende_spalten}")
46     else:
47         print("Alle erforderlichen Spalten sind vorhanden.")
48
49     # 4. Gesamtzahl der Straftaten für ganz Berlin finden
50     gesamt_row = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
51     if gesamt_row.empty:
52         raise ValueError("Die Zeile mit 'Berlin (PKS gesamt)' wurde nicht
53         gefunden.")
54
55     gesamt_straftaten = gesamt_row['Straftaten_insgesamt'].iloc[0]
56     print(f"Gesamtzahl der Straftaten in Berlin: {gesamt_straftaten}")
57
58     if gesamt_straftaten == 0:
59         raise ValueError("Die Gesamtzahl der Straftaten ist 0, kann keine

```

```

Prozente berechnen.")

52
53 # 5. Prozentualen Anteil der einzelnen Bezirke berechnen
54 # Ausschließen der Gesamtzeile von den einzelnen Bezirken
55 einzel_bezirke = df[df['Bezirke'] != 'Berlin (PKS gesamt)'].copy()
56
57 # Berechnung des prozentualen Anteils
58 einzel_bezirke['Prozentualer_Anteil'] = (einzel_bezirke['
Straftaten_insgesamt'] / gesamt_straftaten) * 100
59
60 # Optional: Auf zwei Dezimalstellen runden
61 einzel_bezirke['Prozentualer_Anteil'] = einzel_bezirke['
Prozentualer_Anteil'].round(2)
62
63 # 6. Ergebnis als DataFrame zurückgeben
64 ergebnis = einzel_bezirke[['Bezirke', 'Straftaten_insgesamt', '
Prozentualer_Anteil']]
65
66 return ergebnis
67
68 if __name__ == "__main__":
69     dateipfad = 'Fallzahlen.xlsx' # Pfad zur Excel-Datei anpassen, falls
erforderlich
70     try:
71         ergebnis_df = verarbeite_fallzahlen(dateipfad)
72         print("\nProzentualer Anteil der Straftaten pro Bezirk in Berlin:")
73         print(ergebnis_df.to_string(index=False))
74     except Exception as e:
75         print(f"Ein Fehler ist aufgetreten: {e}")
76
77
78 ### Erklärung der Schritte
79
80 1. **Excel-Datei einlesen:**
81     - Das Skript verwendet 'pd.read_excel', um die Excel-Datei mit dem
angegebenen Sheet-Namen ('Fallzahlen_2023') einzulesen.
82     - Fehler beim Einlesen der Datei werden abgefangen und eine entsprechende
Fehlermeldung ausgegeben.
83
84 2. **Überprüfen der notwendigen Spalten:**
85     - Es werden überprüft, ob die Spalten 'Bezirke' und '
Straftaten_insgesamt' im DataFrame vorhanden sind.
86     - Falls eine oder beide Spalten fehlen, wird eine Fehlermeldung ausgegeben
.
87
88 3. **Gesamtzahl der Straftaten finden:**
89     - Die Zeile mit dem Bezirk 'Berlin (PKS gesamt)' wird gefunden, um die
Gesamtzahl der Straftaten in Berlin zu ermitteln.
90     - Es wird überprüft, ob diese Zeile existiert und ob die Gesamtzahl der
Straftaten nicht null ist.
91
92 4. **Berechnung des prozentualen Anteils:**

```

```

93 - Die Gesamtzeile wird aus dem DataFrame ausgeschlossen, sodass nur die
    einzelnen Bezirke verbleiben.
94 - Für jeden Bezirk wird der prozentuale Anteil der Straftaten berechnet
    und als neue Spalte 'Prozentualer_Anteil' hinzugefügt.
95 - Die Prozentsätze werden auf zwei Dezimalstellen gerundet.
96
97 5. **Ergebnis als DataFrame zurückgeben:**
98 - Das Ergebnis enthält die Spalten 'Bezirke', 'Straftaten_insgesamt'
    und 'Prozentualer_Anteil'.
99 - Im Hauptbereich ('if __name__ == "__main__":') wird das Ergebnis
    ausgegeben.
100
101 ### Beispielhafte Ausgabe
102
103 Nach dem Ausführen des Skripts erhalten Sie eine Ausgabe ähnlich der
    folgenden:
104
105 '''
106 Excel-Datei erfolgreich eingelesen.
107 Alle erforderlichen Spalten sind vorhanden.
108 Gesamtzahl der Straftaten in Berlin: 536695
109
110 Prozentualer Anteil der Straftaten pro Bezirk in Berlin:
111
112      Bezirke  Straftaten_insgesamt  Prozentualer_Anteil
113      Mitte      83611      15.57
114      Friedrichshain-Kreuzberg      55838      10.41
115      Pankow      39164      7.30
116      Charlottenburg-Wilmersdorf      53630      10.00
117      Spandau      25564      4.77
118      Steglitz-Zehlendorf      23719      4.42
119      Tempelhof-Schöneberg      41776      7.78
120      Neukölln      45313      8.46
121      Treptow-Köpenick      26099      4.87
122      Marzahn-Hellersdorf      26649      4.97
123      Lichtenberg      31563      5.88
124      Reinickendorf      27344      5.10
125      Stadtgebiet Berlin, nicht zuzuordnen      56425
126      10.53
127
128 '''
129
130 Dieses Ergebnis zeigt den prozentualen Anteil der Straftaten für jeden Bezirk
    im Vergleich zur Gesamtzahl der Straftaten in Berlin.

```

Listing 107: AI Response Testfall 3 Prompt 3 Ausführung 2

```

1 '''python
2 import pandas as pd
3
4 def calculate_bezirk_straftaten_percentage(excel_file: str, sheet_name: str =
    'Fallzahlen_2023') -> pd.DataFrame:
5     """
6     Liest eine Excel-Datei ein und berechnet den prozentualen Anteil der
7     Straftaten_insgesamt für jeden Bezirk im Vergleich zu ganz Berlin.

```

```

8
9 Parameters:
10 - excel_file: Pfad zur Excel-Datei.
11 - sheet_name: Name des Sheets in der Excel-Datei (standardmäßig '
Fallzahlen_2023').
12
13 Returns:
14 - Ein Pandas DataFrame mit den Bezirken und ihrem prozentualen Anteil.
15 """
16
17 try:
18     # 1. Excel-Datei einlesen
19     df = pd.read_excel(excel_file, sheet_name=sheet_name)
20     print("Excel-Datei erfolgreich eingelesen.")
21 except FileNotFoundError:
22     raise FileNotFoundError(f"Die Datei '{excel_file}' wurde nicht
gefunden.")
23 except Exception as e:
24     raise Exception(f"Fehler beim Einlesen der Excel-Datei: {e}")
25
26 # 2. DataFrame gespeichert (bereits in df)
27
28 # 3. Überprüfen, ob die notwendigen Spalten vorhanden sind
29 required_columns = ['Bezirke', 'Straftaten_insgesamt']
30 missing_columns = [col for col in required_columns if col not in df.
columns]
31 if missing_columns:
32     raise ValueError(f"Die folgenden erforderlichen Spalten fehlen: {'',
'.join(missing_columns)}")
33 print("Alle erforderlichen Spalten sind vorhanden.")
34
35 # 4. Gesamtzahl der Straftaten für ganz Berlin finden
36 total_row = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
37 if total_row.empty:
38     raise ValueError("Die Zeile mit 'Berlin (PKS gesamt)' wurde nicht
gefunden.")
39 total_straftaten = total_row['Straftaten_insgesamt'].values[0]
40 print(f"Gesamtzahl der Straftaten für ganz Berlin: {total_straftaten}")
41
42 # 5. Prozentualen Anteil der einzelnen Bezirke berechnen
43 # Ausschließen der Gesamtzeile aus den Bezirken
44 bezirk_df = df[df['Bezirke'] != 'Berlin (PKS gesamt)'].copy()
45
46 # Berechnung des prozentualen Anteils
47 bezirk_df['Prozentualer_Anteil'] = (bezirk_df['Straftaten_insgesamt'] /
total_straftaten) * 100
48
49 # Optional: Runden auf zwei Dezimalstellen
50 bezirk_df['Prozentualer_Anteil'] = bezirk_df['Prozentualer_Anteil'].round
(2)
51
52 # 6. Ergebnis als DataFrame zurückgeben

```

```

53     result_df = bezirk_df[['Bezirke', 'Straftaten_insgesamt', '
Prozentualer_Anteil']]
54
55     print("Prozentualer Anteil der Straftaten pro Bezirk berechnet.")
56     return result_df
57
58 if __name__ == "__main__":
59     # Pfad zur Excel-Datei
60     excel_file = 'Fallzahlen.xlsx'
61
62     # Berechnung durchführen
63     try:
64         prozent_df = calculate_bezirk_straftaten_percentage(excel_file)
65         print("\nErgebnis:")
66         print(prozent_df)
67     except Exception as e:
68         print(f"Ein Fehler ist aufgetreten: {e}")
69
70
71 ### Erläuterung des Skripts:
72
73 1. **Importieren der notwendigen Bibliothek:**
74 - 'pandas' wird verwendet, um die Excel-Datei zu lesen und die Daten zu
verarbeiten.
75
76 2. **Funktion 'calculate_bezirk_straftaten_percentage':**
77 - **Parameter:**
78 - 'excel_file': Der Name oder Pfad der Excel-Datei (z.B. 'Fallzahlen.
xlsx').
79 - 'sheet_name': Der Name des Sheets in der Excel-Datei, standardmäßig '
Fallzahlen_2023'.
80 - **Schritte innerhalb der Funktion:**
81 - **Einlesen der Excel-Datei:** Mit 'pd.read_excel()' wird das
angegebene Sheet in ein DataFrame geladen.
82 - **Überprüfen der erforderlichen Spalten:** Es wird sichergestellt,
dass sowohl 'Bezirke' als auch 'Straftaten_insgesamt' vorhanden sind.
83 - **Finden der Gesamtzahl der Straftaten:** Die Zeile mit 'Berlin (PKS
gesamt)' wird gefiltert, und der Wert aus 'Straftaten_insgesamt' wird
extrahiert.
84 - **Berechnung des prozentualen Anteils:** Für jeden Bezirk wird der
Anteil der Straftaten an der Gesamtzahl berechnet.
85 - **Erstellen des Ergebnis-DatFrames:** Das Ergebnis enthält die
Bezirke, die Gesamtzahl der Straftaten und den prozentualen Anteil.
86
87 3. **Ausführung des Skripts:**
88 - Wenn das Skript direkt ausgeführt wird ('__main__'), wird die Funktion
aufgerufen und das Ergebnis wird ausgegeben.
89 - **Fehlerbehandlung:** Es werden Fehlermeldungen ausgegeben, falls die
Datei nicht gefunden wird oder erforderliche Spalten fehlen.
90
91 ### Beispiel für die Ausgabe:
92

```

```

93 Nach dem Ausführen des Skripts erhalten Sie eine Ausgabe ähnlich der
    folgenden:
94
95 '''
96 Excel-Datei erfolgreich eingelesen.
97 Alle erforderlichen Spalten sind vorhanden.
98 Gesamtzahl der Straftaten für ganz Berlin: 536695
99 Prozentualer Anteil der Straftaten pro Bezirk berechnet.
100
101 Ergebnis:
102
103      Bezirke  Straftaten_insgesamt  Prozentualer_Anteil
104 0           Mitte                83611                15.58
105 1   Friedrichshain-Kreuzberg        55838                10.40
106 2           Pankow                39164                 7.30
107 3   Charlottenburg-Wilmersdorf       53630                10.00
108 4           Spandau               25564                 4.77
109 5   Steglitz-Zehlendorf             23719                 4.42
110 6   Tempelhof-Schöneberg           41776                 7.78
111 7           Neukölln              45313                 8.46
112 8   Treptow-Köpenick               26099                 4.87
113 9   Marzahn-Hellersdorf            26649                 4.97
114 10          Lichtenberg            31563                 5.89
115 11          Reinickendorf          27344                 5.10
116 12 Stadtgebiet Berlin, nicht zuzuordnen 56425
    10.52
117
118 '''
119
120 Dieses Ergebnis zeigt für jeden Bezirk die Gesamtzahl der Straftaten sowie
    den prozentualen Anteil an den gesamten Straftaten in Berlin.

```

Listing 108: AI Response Testfall 3 Prompt 3 Ausführung 3

```

1 Hier ist ein vollständiges Python-Skript, das die von dir beschriebenen
    Anforderungen erfüllt. Das Skript verwendet die Bibliothek 'pandas', um
    die Excel-Datei zu verarbeiten. Stelle sicher, dass du 'pandas'
    installiert hast ('pip install pandas openpyxl'), bevor du das Skript ausf
    ührst.
2
3 '''python
4 import pandas as pd
5
6 def main():
7     # 1. Die Excel-Datei einlesen
8     try:
9         df = pd.read_excel('Fallzahlen.xlsx', sheet_name='Fallzahlen_2023')
10    except FileNotFoundError:
11        print("Die Datei 'Fallzahlen.xlsx' wurde nicht gefunden.")
12        return
13    except Exception as e:
14        print(f"Ein Fehler ist beim Einlesen der Excel-Datei aufgetreten: {e}
15    ")
16        return

```



```

17     # 2. Die Tabelle als Pandas DataFrame gespeichert (bereits in 'df')
18
19     # 3. Überprüfen, ob die notwendigen Spalten vorhanden sind
20     notwendige_spalten = ['Bezirke', 'Straftaten_insgesamt']
21     fehlende_spalten = [spalte for spalte in notwendige_spalten wenn spalte
22                          nicht in df.columns]
23     wenn fehlende_spalten:
24         print(f"Die folgenden notwendigen Spalten fehlen: {'', '.join(
25               fehlende_spalten)}")
26         return
27
28     # 4. Finde die Gesamtzahl der Straftaten für ganz Berlin
29     gesamt_row = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
30     wenn gesamt_row.empty:
31         print("Die Zeile mit Bezirk 'Berlin (PKS gesamt)' wurde nicht
32               gefunden.")
33         return
34     gesamt_straftaten = gesamt_row['Straftaten_insgesamt'].values[0]
35
36     # 5. Berechne den prozentualen Anteil der einzelnen Bezirke von ganz
37     Berlin
38     # Exkludiere die Gesamtzeile aus der Berechnung
39     bezirke_df = df[df['Bezirke'] != 'Berlin (PKS gesamt)'].copy()
40     bezirke_df['Prozentualer_Anteil'] = (bezirke_df['Straftaten_insgesamt'] /
41                                         gesamt_straftaten) * 100
42
43     # Optional: Runden auf zwei Dezimalstellen
44     bezirke_df['Prozentualer_Anteil'] = bezirke_df['Prozentualer_Anteil'].
45     round(2)
46
47     # 6. Das Ergebnis als DataFrame zurückgeben
48     ergebnis_df = bezirke_df[['Bezirke', 'Straftaten_insgesamt', '
49                               Prozentualer_Anteil']]
50
51     # Ergebnis anzeigen
52     print(ergebnis_df)
53
54     if __name__ == "__main__":
55         main()
56
57     """
58
59     ### Erklärung der Schritte:
60
61     1. **Excel-Datei einlesen**:
62         - Das Skript versucht, die Excel-Datei 'Fallzahlen.xlsx' aus dem Sheet '
63           Fallzahlen_2023' zu lesen.
64         - Fehler beim Einlesen werden abgefangen und entsprechende Meldungen
65           ausgegeben.
66
67     2. **Überprüfung der notwendigen Spalten**:
68         - Es wird geprüft, ob die Spalten 'Bezirke' und 'Straftaten_insgesamt' im
69           DataFrame vorhanden sind.

```

```

59 - Falls eine der Spalten fehlt, wird eine Fehlermeldung ausgegeben und das
    Skript beendet.
60
61 3. **Gesamtzahl der Straftaten bestimmen**:
62 - Die Zeile mit dem Bezirk 'Berlin (PKS gesamt)' wird gesucht, um die
    Gesamtzahl der Straftaten in ganz Berlin zu erhalten.
63 - Falls diese Zeile nicht gefunden wird, wird eine Fehlermeldung
    ausgegeben.
64
65 4. **Prozentualen Anteil berechnen**:
66 - Die Gesamtzeile wird aus dem DataFrame entfernt, um nur die einzelnen
    Bezirke zu betrachten.
67 - Für jeden Bezirk wird der prozentuale Anteil der Straftaten am
    Gesamtwert berechnet.
68 - Optional wird der prozentuale Anteil auf zwei Dezimalstellen gerundet.
69
70 5. **Ergebnis darstellen**:
71 - Es wird ein neues DataFrame erstellt, das die Bezirke, die Gesamtzahl
    der Straftaten und den prozentualen Anteil enthält.
72 - Das Ergebnis wird in der Konsole ausgegeben.
73
74 ### Beispielausgabe:
75
76 '''
77
78           Bezirke  Straftaten_insgesamt  Prozentualer_Anteil
79 0           Mitte                83611                15.60
80 1  Friedrichshain-Kreuzberg                55838                10.40
81 2           Pankow                39164                 7.30
82 3  Charlottenburg-Wilmersdorf                53630                10.00
83 4           Spandau                25564                 4.80
84 5  Steglitz-Zehlendorf                23719                 4.40
85 6  Tempelhof-Schöneberg                41776                 7.80
86 7           Neukölln                45313                 8.50
87 8  Treptow-Köpenick                26099                 4.90
88 9  Marzahn-Hellersdorf                26649                 5.00
89 10          Lichtenberg                31563                 5.90
90 11  Reinickendorf                27344                 5.10
91 12 Stadtgebiet Berlin, nicht zuzuordnen                56425
92    10.50
93 '''
94
95 Dieses Skript sollte genau das tun, was du benötigst. Passe gegebenenfalls
    den Pfad zur Excel-Datei oder den Sheet-Namen an, falls diese abweichen.

```

Listing 109: AI Response Testfall 3 Prompt 3 Ausführung 4

```

1 Hier ist ein Python-Skript, das die von Ihnen beschriebenen Anforderungen erf
  üllt. Das Skript verwendet die 'pandas'-Bibliothek, um die Excel-Datei zu
  lesen und die notwendigen Berechnungen durchzuführen.
2
3 '''python
4 import pandas as pd
5

```

```

6 def berechne_prozentanteile(dateipfad: str, sheet_name: str = '
Fallzahlen_2023') -> pd.DataFrame:
7     """
8     Liest eine Excel-Datei ein, überprüft notwendige Spalten, berechnet den
    prozentualen Anteil
9     der Straftaten pro Bezirk und gibt das Ergebnis als DataFrame zurück.
10
11     :param dateipfad: Pfad zur Excel-Datei (z.B. 'Fallzahlen.xlsx')
12     :param sheet_name: Name des Sheets in der Excel-Datei (Standard: '
Fallzahlen_2023')
13     :return: DataFrame mit Bezirken und deren prozentualen Anteil an den
    Straftaten
14     """
15     # 1. Excel-Datei einlesen
16     try:
17         df = pd.read_excel(dateipfad, sheet_name=sheet_name)
18         print(f"Excel-Datei '{dateipfad}' erfolgreich eingelesen.")
19     except FileNotFoundError:
20         print(f"Fehler: Die Datei '{dateipfad}' wurde nicht gefunden.")
21         return
22     except Exception as e:
23         print(f"Ein Fehler ist beim Einlesen der Excel-Datei aufgetreten: {e}
    ")
24         return
25
26     # 2. DataFrame speichern (bereits durch pd.read_excel erledigt)
27
28     # 3. Überprüfen, ob die notwendigen Spalten vorhanden sind
29     notwendige_spalten = {'Bezirke', 'Straftaten_insgesamt'}
30     vorhandene_spalten = set(df.columns)
31
32     fehlende_spalten = notwendige_spalten - vorhandene_spalten
33     if fehlende_spalten:
34         print(f"Fehler: Die folgenden notwendigen Spalten fehlen in der
    Tabelle: {fehlende_spalten}")
35         return
36     else:
37         print("Überprüfung der notwendigen Spalten erfolgreich.")
38
39     # 4. Gesamtzahl der Straftaten für ganz Berlin finden
40     gesamt_row = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
41     if gesamt_row.empty:
42         print("Fehler: Keine Zeile mit 'Bezirke' als 'Berlin (PKS gesamt)'
    gefunden.")
43         return
44
45     gesamt_straftaten = gesamt_row['Straftaten_insgesamt'].iloc[0]
46     print(f"Gesamtzahl der Straftaten in Berlin: {gesamt_straftaten}")
47
48     # 5. Prozentualen Anteil der einzelnen Bezirke berechnen
49     # Ausschließen der Gesamtzeile aus den Bezirken
50     bezirke_df = df[df['Bezirke'] != 'Berlin (PKS gesamt)'].copy()

```

```

51
52 # Berechnung des prozentualen Anteils
53 bezirke_df['Prozentualer_Anteil'] = (bezirke_df['Straftaten_insgesamt'] /
54     gesamt_straftaten) * 100
55
56 # Optional: Auf zwei Dezimalstellen runden
57 bezirke_df['Prozentualer_Anteil'] = bezirke_df['Prozentualer_Anteil'].
58     round(2)
59
60 # 6. Ergebnis als DataFrame zurückgeben
61 ergebnis_df = bezirke_df[['Bezirke', 'Straftaten_insgesamt', '
62     Prozentualer_Anteil']]
63
64 return ergebnis_df
65
66 # Beispielhafte Nutzung des Skripts
67 if __name__ == "__main__":
68     dateipfad = 'Fallzahlen.xlsx' # Pfad zur Excel-Datei
69     ergebnis = berechne_prozentanteile(dateipfad)
70
71     if ergebnis is not None:
72         print("\nProzentuale Anteile der Straftaten pro Bezirk:")
73         print(ergebnis)
74
75         # Optional: Ergebnis in eine neue Excel-Datei speichern
76         # ergebnis.to_excel('Prozentuale_Anteile_Straftaten.xlsx', index=
77             False)
78
79 '''
80
81 ### Erklärung der Schritte:
82
83 1. **Excel-Datei einlesen:**
84 - Das Skript verwendet 'pd.read_excel', um die Excel-Datei einzulesen.
85 Standardmäßig wird das Sheet 'Fallzahlen_2023' gelesen, es kann jedoch
86 ein anderes Sheet angegeben werden.
87
88 2. **Überprüfung der notwendigen Spalten:**
89 - Es wird überprüft, ob die Spalten 'Bezirke' und 'Straftaten_insgesamt'
90 im DataFrame vorhanden sind. Falls nicht, wird eine Fehlermeldung
91 ausgegeben und das Skript beendet.
92
93 3. **Gesamtzahl der Straftaten ermitteln:**
94 - Die Gesamtzahl der Straftaten für ganz Berlin wird aus der Zeile mit '
95 Bezirke' gleich 'Berlin (PKS gesamt)' entnommen.
96
97 4. **Berechnung des prozentualen Anteils:**
98 - Die Gesamtzeile wird aus den Bezirken ausgeschlossen, um eine korrekte
99 Prozentberechnung zu gewährleisten.
100 - Der prozentuale Anteil wird für jeden Bezirk berechnet und als neue
101 Spalte 'Prozentualer_Anteil' zum DataFrame hinzugefügt.
102
103 5. **Ergebnis zurückgeben:**

```

```

92 - Das Ergebnis wird als neuer DataFrame mit den Spalten 'Bezirke', '
Straftaten_insgesamt' und 'Prozentualer_Anteil' zurückgegeben und im
Hauptteil des Skripts ausgegeben.
93
94 ### Hinweise:
95
96 - **Fehlerbehandlung:** Das Skript enthält grundlegende Fehlerbehandlungen, z
.B. wenn die Datei nicht gefunden wird oder notwendige Spalten fehlen.
97
98 - **Speichern des Ergebnisses:** Optional können Sie das Ergebnis in eine
neue Excel-Datei speichern, indem Sie die auskommentierte 'to_excel'-Zeile
aktivieren.
99
100 - **Anpassungen:** Stellen Sie sicher, dass der Pfad zur Excel-Datei ('
dateipfad') korrekt ist oder passen Sie den Skriptaufruf entsprechend an.
101
102 ### Voraussetzungen:
103
104 - **Installation von pandas:**
105 Stellen Sie sicher, dass die 'pandas'-Bibliothek installiert ist. Falls
nicht, können Sie sie mit folgendem Befehl installieren:
106
107 '''bash
108 pip install pandas
109 '''
110
111 - **Installation von openpyxl:**
112 Für das Einlesen von Excel-Dateien benötigen Sie möglicherweise das '
openpyxl'-Paket:
113
114 '''bash
115 pip install openpyxl
116 '''
117
118 Führen Sie das Skript aus Ihrer Python-Umgebung aus, um die gewünschten
Berechnungen durchzuführen.

```

Listing 110: AI Response Testfall 3 Prompt 3 Ausführung 5

```

1 Hier ist ein vollständiges Python-Skript, das die beschriebenen Anforderungen
erfüllt. Das Skript liest die Excel-Datei 'Fallzahlen.xlsx', verarbeitet
alle Sheets, extrahiert die Gesamtzahl der Straftaten für Berlin,
berechnet die prozentuale Veränderung zum Vorjahr und erstellt schließlich
eine Zeitreihe als Pandas DataFrame.
2
3 '''python
4 import pandas as pd
5
6 def extract_year(sheet_name):
7     """
8     Extrahiert das Jahr aus dem Sheetnamen.
9     Annahme: Der Sheetname enthält das Jahr als vierstellige Zahl, z.B. "2020
", "2021".

```

```

10     """
11     import re
12     match = re.search(r'(\d{4})', sheet_name)
13     if match:
14         return int(match.group(1))
15     else:
16         raise ValueError(f"Kein Jahr im Sheetnamen '{sheet_name}' gefunden.")
17
18 def get_total_crimes(df):
19     """
20     Extrahiert die Gesamtzahl der Straftaten für Berlin aus dem DataFrame.
21     Annahme: Die Zeile mit 'Bezirke' == 'Berlin (PKS gesamt)' enthält die
22     Gesamtzahl.
23     """
24     total_row = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
25     if total_row.empty:
26         raise ValueError("Keine Zeile mit 'Berlin (PKS gesamt)' gefunden.")
27     return total_row['Straftaten_insgesamt'].values[0]
28
29 def main():
30     # Pfad zur Excel-Datei
31     excel_file = 'Fallzahlen.xlsx'
32
33     # Lade die Excel-Datei
34     try:
35         xls = pd.ExcelFile(excel_file)
36     except FileNotFoundError:
37         print(f"Datei '{excel_file}' wurde nicht gefunden.")
38         return
39     except Exception as e:
40         print(f"Fehler beim Laden der Datei: {e}")
41         return
42
43     # Liste der Sheets
44     sheet_names = xls.sheet_names
45
46     # Liste zur Speicherung der Ergebnisse
47     data = []
48
49     for sheet in sheet_names:
50         try:
51             # Lade das Sheet als DataFrame
52             df = pd.read_excel(xls, sheet_name=sheet)
53
54             # Extrahiere das Jahr
55             year = extract_year(sheet)
56
57             # Extrahiere die Gesamtzahl der Straftaten
58             total_crimes = get_total_crimes(df)
59
60             # Speichere das Ergebnis
61             data.append({'Jahr': year, 'Straftaten_insgesamt': total_crimes})

```

```

61         except Exception as e:
62             print(f"Fehler beim Verarbeiten des Sheets '{sheet}': {e}")
63             continue
64
65
66     # Erstelle einen DataFrame aus den gesammelten Daten
67     crimes_df = pd.DataFrame(data)
68
69     # Sortiere den DataFrame nach Jahr
70     crimes_df.sort_values('Jahr', inplace=True)
71     crimes_df.reset_index(drop=True, inplace=True)
72
73     # Berechne die prozentuale Veränderung zum Vorjahr
74     crimes_df['Prozentuale_Veraenderung'] = crimes_df['Straftaten_insgesamt']
75     ].pct_change() * 100
76
77     # Optional: Runde die Prozentwerte auf zwei Dezimalstellen
78     crimes_df['Prozentuale_Veraenderung'] = crimes_df['
79     Prozentuale_Veraenderung'].round(2)
80
81     # Setze das Jahr als Index (optional)
82     crimes_df.set_index('Jahr', inplace=True)
83
84     # Anzeige des resultierenden DataFrames
85     print(crimes_df)
86
87     # Optional: Speichern des DataFrames als CSV
88     # crimes_df.to_csv('Prozentuale_Veraenderung_Straftaten_Berlin.csv')
89
90 if __name__ == "__main__":
91     main()
92
93
94 ### Erklärung des Skripts:
95
96 1. **Import der notwendigen Bibliotheken:**
97     - 'pandas' wird für die Datenverarbeitung verwendet.
98     - 're' (Regular Expressions) wird zur Extraktion des Jahres aus dem
99     Sheetnamen genutzt.
100
101 2. **Funktionen:**
102     - 'extract_year(sheet_name)': Diese Funktion sucht nach einem
103     vierstelligen Jahr im Sheetnamen. Wenn kein Jahr gefunden wird, wird eine
104     Fehlermeldung ausgegeben.
105     - 'get_total_crimes(df)': Diese Funktion filtert die Zeile, in der die
106     Spalte 'Bezirke' den Wert 'Berlin (PKS gesamt)' hat, und gibt den Wert
107     aus der Spalte 'Straftaten_insgesamt' zurück.
108
109 3. **Hauptfunktion ('main'):**
110     - Lädt die Excel-Datei 'Fallzahlen.xlsx'.
111     - Iteriert über alle Sheets in der Excel-Datei:
112         - Liest jedes Sheet als DataFrame.

```

```

106 - Extrahiert das Jahr aus dem Sheetnamen.
107 - Extrahiert die Gesamtzahl der Straftaten für Berlin.
108 - Speichert das Jahr und die Straftaten in einer Liste.
109 - Erstellt einen finalen DataFrame 'crimes_df' aus den gesammelten Daten.
110 - Sortiert den DataFrame nach Jahr.
111 - Berechnet die prozentuale Veränderung der Straftaten zum Vorjahr.
112 - Setzt das Jahr als Index des DataFrames.
113 - Gibt den resultierenden DataFrame aus.
114 - Optional kann der DataFrame als CSV-Datei gespeichert werden (die
    entsprechende Zeile ist auskommentiert).
115
116 ### Annahmen:
117 - **Sheetnamen enthalten das Jahr:** Das Skript geht davon aus, dass die
    Sheetnamen ein vierstelliges Jahr enthalten, z.B. "2020", "2021". Falls
    dies nicht der Fall ist, muss die Funktion 'extract_year' entsprechend
    angepasst werden.
118 - **Struktur der Sheets:** Jede Sheet enthält eine Spalte 'Bezirke' und eine
    Spalte 'Straftaten_insgesamt', und es gibt eine Zeile mit 'Bezirke' gleich
    'Berlin (PKS gesamt)'.
119 - **Fehlerbehandlung:** Das Skript enthält grundlegende Fehlerbehandlungen,
    um Probleme beim Laden der Datei oder beim Verarbeiten einzelner Sheets zu
    erkennen und anzuzeigen, ohne das gesamte Skript zu stoppen.
120
121 ### Hinweise:
122 - **Anpassung der Sheetnamen-Logik:** Falls die Sheetnamen anders
    strukturiert sind oder zusätzliche Informationen enthalten, muss die '
    extract_year'-Funktion entsprechend angepasst werden.
123 - **Erweiterung des Skripts:** Das Skript kann erweitert werden, um weitere
    Analysen durchzuführen oder die Ergebnisse in verschiedenen Formaten zu
    speichern.
124
125 ### Ausführung:
126 Stellen Sie sicher, dass die Datei 'Fallzahlen.xlsx' im gleichen Verzeichnis
    wie das Skript liegt oder passen Sie den Pfad zur Datei entsprechend an. F
    ühren Sie dann das Skript mit Python aus:
127
128 ```bash
129 python dein_skriptname.py
130 ```
131
132 Das Skript gibt einen DataFrame aus, der die Jahre, die Gesamtzahl der
    Straftaten und die prozentuale Veränderung zum Vorjahr enthält.

```

Listing 111: AI Response Testfall 4 Prompt 1 Ausführung 1

```

1 Natürlich! Unten findest du ein Python-Skript, das deine Anforderungen erfü
    llt. Dieses Skript verwendet die 'pandas'-Bibliothek, um die Excel-Datei
    zu lesen, die relevanten Daten aus allen Sheets zu extrahieren, die jäh
    rliche Gesamtstrafstatenzahl für ganz Berlin zu berechnen und schließlich
    die prozentuale Veränderung zum Vorjahr darzustellen.
2
3 Stelle sicher, dass du die notwendigen Bibliotheken installiert hast. Du
    kannst dies mit 'pip' tun, falls sie noch nicht installiert sind:

```



```

4
5 '''bash
6 pip install pandas openpyxl
7 '''
8
9 ### Python-Skript: 'analyse_fallzahlen.py'
10
11 '''python
12 import pandas as pd
13
14 def extract_berlin_total(df):
15     """
16     Extrahiert die Gesamtzahl der Straftaten für Berlin aus dem DataFrame.
17
18     Es wird entweder nach dem 'LOR-Schlüssel' 999999 oder nach dem
19     Bezirksnamen
20     'Berlin (PKS gesamt)' gesucht.
21
22     Args:
23         df (pd.DataFrame): Der DataFrame des aktuellen Sheets.
24
25     Returns:
26         int oder float: Die Gesamtzahl der Straftaten in Berlin.
27     """
28     # Versuch 1: Nach LOR-Schlüssel 999999 filtern
29     berlin_row = df[df['LOR-Schlüssel'] == 999999]
30
31     if not berlin_row.empty:
32         return berlin_row['Straftaten_insgesamt'].values[0]
33
34     # Versuch 2: Nach Bezirksnamen 'Berlin (PKS gesamt)' filtern
35     berlin_row = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
36
37     if not berlin_row.empty:
38         return berlin_row['Straftaten_insgesamt'].values[0]
39
40     # Wenn nichts gefunden wurde, gebe NaN zurück
41     return float('nan')
42
43 def main():
44     # Pfad zur Excel-Datei
45     excel_file = 'Fallzahlen.xlsx'
46
47     try:
48         # Lade alle Sheets der Excel-Datei
49         xls = pd.ExcelFile(excel_file)
50     except FileNotFoundError:
51         print(f"Die Datei {excel_file} wurde nicht gefunden.")
52         return
53     except Exception as e:
54         print(f"Fehler beim Laden der Excel-Datei: {e}")
55         return

```

```

55
56 # Liste zur Speicherung der Daten
57 data = []
58
59 for sheet_name in xls.sheet_names:
60     try:
61         # Lese das aktuelle Sheet
62         df = pd.read_excel(xls, sheet_name=sheet_name)
63
64         # Extrahiere die Gesamtzahl der Straftaten für Berlin
65         total_crimes = extract_berlin_total(df)
66
67         # Versuche, das Jahr aus dem Sheet-Namen zu extrahieren
68         try:
69             year = int(sheet_name)
70         except ValueError:
71             # Wenn der Sheet-Name kein Jahr ist, überspringe dieses Sheet
72             print(f"Sheet '{sheet_name}' entspricht nicht dem erwarteten
Jahresformat und wird übersprungen.")
73             continue
74
75         # Füge die Daten zur Liste hinzu
76         data.append({'Jahr': year, 'Straftaten_insgesamt': total_crimes})
77
78     except Exception as e:
79         print(f"Fehler beim Verarbeiten des Sheets '{sheet_name}': {e}")
80         continue
81
82 # Erstelle einen DataFrame aus den gesammelten Daten
83 df_totals = pd.DataFrame(data)
84
85 if df_totals.empty:
86     print("Keine gültigen Daten gefunden.")
87     return
88
89 # Sortiere den DataFrame nach Jahr
90 df_totals.sort_values('Jahr', inplace=True)
91
92 # Setze das Jahr als Index
93 df_totals.set_index('Jahr', inplace=True)
94
95 # Berechne die prozentuale Veränderung zum Vorjahr
96 df_totals['Prozentuale_Veraenderung'] = df_totals['Straftaten_insgesamt']
.pct_change() * 100
97
98 # Optional: Runde die Prozentwerte auf zwei Dezimalstellen
99 df_totals['Prozentuale_Veraenderung'] = df_totals['
Prozentuale_Veraenderung'].round(2)
100
101 # Anzeige des resultierenden DataFrames
102 print(df_totals)
103

```

```

104 # Optional: Speichere das Ergebnis in eine neue Excel-Datei
105 output_file = 'Straftaten_Zeitreihe.xlsx'
106 try:
107     df_totals.to_excel(output_file)
108     print(f"Das Ergebnis wurde erfolgreich in '{output_file}' gespeichert
109     .")
110 except Exception as e:
111     print(f"Fehler beim Speichern der Ergebnisdatei: {e}")
112
113 if __name__ == "__main__":
114     main()
115
116 ### Erklärung des Skripts
117
118 1. **Importieren der Bibliotheken**:
119     - 'pandas' wird für die Datenmanipulation verwendet.
120
121 2. **Funktion 'extract_berlin_total'**:
122     - Diese Funktion durchsucht den gegebenen DataFrame nach der Gesamtzahl
123     der Straftaten für ganz Berlin.
124     - Sie sucht zuerst nach dem 'LOR-Schlüssel' '999999'. Wenn dieser nicht
125     gefunden wird, sucht sie nach dem Bezirksnamen 'Berlin (PKS gesamt)'.
126     - Wenn beide Suchen fehlschlagen, gibt sie 'NaN' zurück.
127
128 3. **Funktion 'main'**:
129     - Lädt die Excel-Datei und iteriert über alle Sheets.
130     - Extrahiert für jedes Sheet die Gesamtzahl der Straftaten in Berlin.
131     - Versucht, das Jahr aus dem Sheet-Namen zu extrahieren. Wenn der Sheet-
132     Name kein gültiges Jahr ist, wird das Sheet übersprungen.
133     - Speichert die gesammelten Daten in einem DataFrame.
134     - Sortiert die Daten nach Jahr und berechnet die prozentuale Veränderung
135     zum Vorjahr.
136     - Gibt den resultierenden DataFrame aus und speichert ihn optional in
137     eine neue Excel-Datei namens 'Straftaten_Zeitreihe.xlsx'.
138
139 4. **Ausführen des Skripts**:
140     - Das Skript kann direkt ausgeführt werden, indem du es speicherst (z.B.
141     als 'analyse_fallzahlen.py') und über die Kommandozeile startest:
142     ```bash
143     python analyse_fallzahlen.py
144     ```
145
146 ### Beispielausgabe
147
148 Nach dem Ausführen des Skripts erhältst du einen DataFrame ähnlich dem
149 folgenden:
150
151 '''
152 Straftaten_insgesamt    Prozentuale_Veraenderung
153 Jahr
154 2018                    500000                    NaN

```

```

148 2019          520000          4.00
149 2020          510000         -1.92
150 2021          530000          3.92
151 2022          550000          3.77
152 2023          580000          5.45
153 '''
154
155 Hierbei zeigt die Spalte 'Prozentuale_Veraenderung' die jährliche Veränderung
    der Gesamtstraftaten in Berlin im Vergleich zum Vorjahr.
156
157 ### Anmerkungen
158
159 - **Sheet-Namen**: Das Skript geht davon aus, dass die Sheet-Namen ausschließ-
    lich aus dem Jahr bestehen (z.B. '2020', '2021', etc.). Wenn deine
    Sheet-Namen ein anderes Format haben (z.B. 'Daten_2020'), musst du die
    Logik zur Extraktion des Jahres entsprechend anpassen.
160
161 - **Fehlerbehandlung**: Das Skript enthält grundlegende Fehlerbehandlungen,
    um sicherzustellen, dass fehlende Dateien oder unerwartete Sheet-Namen
    nicht zu einem Absturz führen, sondern entsprechende Meldungen ausgeben.
162
163 - **Anpassungen**: Je nach genauer Struktur deiner Excel-Datei und den Sheet-
    Namen könnte es notwendig sein, kleinere Anpassungen am Skript vorzunehmen
    .
164
165 Ich hoffe, dieses Skript hilft dir bei der Analyse deiner Fallzahlen! Wenn du
    weitere Fragen hast oder Anpassungen benötigst, lass es mich wissen.

```

Listing 112: AI Response Testfall 4 Prompt 1 Ausführung 2

```

1 Hier ist ein Python-Skript, das deine Anforderungen erfüllt. Das Skript liest
    alle Sheets der Excel-Datei 'Fallzahlen.xlsx', extrahiert die Gesamtzahl
    der Straftaten für Berlin aus jedem Sheet, berechnet die prozentuale Verä-
    2 nderung zum Vorjahr und erstellt einen Pandas DataFrame mit der Zeitreihe.
3
4 '''python
5
6 import pandas as pd
7
8 # Pfad zur Excel-Datei
9 excel_datei = 'Fallzahlen.xlsx'
10
11 # Lade alle Sheets in ein Dictionary
12 alle_sheets = pd.read_excel(excel_datei, sheet_name=None)
13
14 # Liste zur Speicherung der jährlichen Straftaten
15 jahrliche_straftaten = []
16
17 for sheet_name, df in alle_sheets.items():
18     try:
19         # Versuche, das Jahr aus dem Sheetnamen zu extrahieren
20         jahr = int(sheet_name)
21     except ValueError:
22         # Wenn der Sheetname kein Jahr ist, überspringe dieses Sheet

```

```

21     print(f"Sheet '{sheet_name}' übersprungen (kein gültiges Jahr).")
22     continue
23
24     # Finde die Zeile für 'Berlin (PKS gesamt)'
25     berlin_gesamt = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
26
27     if nicht berlin_gesamt.empty:
28         # Extrahiere die Gesamtzahl der Straftaten
29         gesamt_straftaten = berlin_gesamt['Straftaten_insgesamt'].values[0]
30         jährliche_straftaten.append({'Jahr': jahr, 'Straftaten_insgesamt':
gesamt_straftaten})
31     else:
32         print(f"Keine Daten für 'Berlin (PKS gesamt)' in Sheet '{sheet_name}'
gefunden.")
33
34     # Erstelle einen DataFrame aus den gesammelten Daten
35     df_jährlich = pd.DataFrame(jährliche_straftaten)
36
37     # Sortiere den DataFrame nach Jahr
38     df_jährlich = df_jährlich.sort_values('Jahr').reset_index(drop=True)
39
40     # Berechne die prozentuale Veränderung zum Vorjahr
41     df_jährlich['Veränderung_prozent'] = df_jährlich['Straftaten_insgesamt'].
pct_change() * 100
42
43     # Runde die Veränderungen auf zwei Dezimalstellen
44     df_jährlich['Veränderung_prozent'] = df_jährlich['Veränderung_prozent'].round
(2)
45
46     # Anzeige des resultierenden DataFrames
47     print(df_jährlich)
48     '''
49
50     ### Erklärung des Skripts
51
52     1. **Import der Bibliotheken**:
53         '''python
54         import pandas as pd
55         '''
56         Wir verwenden 'pandas' für die Datenverarbeitung.
57
58     2. **Laden der Excel-Datei**:
59         '''python
60         excel_datei = 'Fallzahlen.xlsx'
61         alle_sheets = pd.read_excel(excel_datei, sheet_name=None)
62         '''
63         Mit 'sheet_name=None' werden alle Sheets in ein Dictionary geladen, wobei
die Schlüssel die Sheetnamen sind.
64
65     3. **Extrahieren der jährlichen Straftaten**:
66         '''python
67         for sheet_name, df in alle_sheets.items():

```

```

68     try:
69         jahr = int(sheet_name)
70     except ValueError:
71         print(f"Sheet '{sheet_name}' übersprungen (kein gültiges Jahr).")
72         continue
73
74     berlin_gesamt = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
75
76     if not berlin_gesamt.empty:
77         gesamt_straftaten = berlin_gesamt['Straftaten_insgesamt'].values
78         [0]
79         jährliche_straftaten.append({'Jahr': jahr, 'Straftaten_insgesamt':
80 : gesamt_straftaten})
81     else:
82         print(f"Keine Daten für 'Berlin (PKS gesamt)' in Sheet '{
83 sheet_name}' gefunden.")
84     '''
85     - Es wird angenommen, dass jeder Sheetname ein Jahr darstellt (z.B. '2020
86 ', '2021', etc.).
87     - Das Skript sucht in jedem Sheet nach der Zeile, in der die Bezirke '
88 Berlin (PKS gesamt)' sind, und extrahiert die Gesamtzahl der Straftaten.
89
90 4. **Erstellen des DataFrames**:
91     '''python
92     df_jährlich = pd.DataFrame(jährliche_straftaten)
93     df_jährlich = df_jährlich.sort_values('Jahr').reset_index(drop=True)
94     '''
95     Die gesammelten Daten werden in einen DataFrame umgewandelt und nach Jahr
96     sortiert.
97
98 5. **Berechnung der prozentualen Veränderung**:
99     '''python
100     df_jährlich['Veränderung_prozent'] = df_jährlich['Straftaten_insgesamt'].
101     pct_change() * 100
102     df_jährlich['Veränderung_prozent'] = df_jährlich['Veränderung_prozent'].
103     round(2)
104     '''
105     - 'pct_change()' berechnet die prozentuale Veränderung zum Vorjahr.
106     - Die Ergebnisse werden auf zwei Dezimalstellen gerundet.
107
108 6. **Ausgabe des Ergebnisses**:
109     '''python
110     print(df_jährlich)
111     '''
112     Dies gibt den finalen DataFrame mit den jährlichen Straftaten und der
113     prozentualen Veränderung zum Vorjahr aus.
114
115 ### Beispielhafte Ausgabe
116
117 Angenommen, deine Excel-Datei enthält die Daten für die Jahre 2018 bis 2022,
118 könnte die Ausgabe wie folgt aussehen:
119

```

```

110 '''
111     Jahr   Straftaten_insgesamt   Veränderung_prozent
112 0   2018                50000                NaN
113 1   2019                52000                4.00
114 2   2020                51000               -1.92
115 3   2021                53000                3.92
116 4   2022                55000                3.77
117 '''
118
119 - **Hinweis**: Für das erste Jahr ('2018' in diesem Beispiel) gibt es keine
    Veränderung zum Vorjahr, daher ist der Wert 'NaN'.
120
121 ### Anpassungen
122
123 Falls deine Sheets nicht direkt das Jahr im Namen enthalten, sondern ein
    anderes Format haben (z.B. 'Daten_2020'), musst du die Methode zur
    Extraktion des Jahres aus dem Sheetnamen anpassen. Hier ein Beispiel, wie
    du das Jahr extrahieren könntest, wenn der Sheetname das Jahr enthält:
124
125 '''python
126 import re
127
128 # Beispiel: Sheetname ist 'Daten_2020'
129 match = re.search(r'(\d{4})', sheet_name)
130 if match:
131     jahr = int(match.group(1))
132 else:
133     print(f"Jahr in Sheet '{sheet_name}' nicht gefunden.")
134     continue
135 '''
136
137 Dieses Snippet verwendet reguläre Ausdrücke, um eine vierstellige Jahreszahl
    im Sheetnamen zu finden.
138
139 ### Fehlerbehandlung
140
141 Das Skript enthält grundlegende Fehlerbehandlungen, wie das Überspringen von
    Sheets, die kein gültiges Jahr als Namen haben, und das Informieren, wenn
    die Zeile für 'Berlin (PKS gesamt)' nicht gefunden wird. Du kannst diese
    erweitern, um spezifischere Anforderungen oder Datenstrukturen zu unterstüt-
    tzen.
142
143 ### Abhängigkeiten
144
145 Stelle sicher, dass du die erforderlichen Python-Pakete installiert hast. Du
    kannst 'pandas' mit folgendem Befehl installieren:
146
147 '''bash
148 pip install pandas
149 '''
150
151 ### Ausführung

```

```

152
153 Speichere das Skript in einer '.py'-Datei, z.B. 'analyse_straftaten.py', und
    führe es in deinem Terminal oder deiner Kommandozeile aus:
154
155 '''bash
156 python analyse_straftaten.py
157 '''
158
159 Dieses Skript sollte dir eine klare Übersicht über die jährlichen Verä
    nderungen der Straftaten in Berlin geben.

```

Listing 113: AI Response Testfall 4 Prompt 1 Ausführung 3

```

1 Natürlich! Hier ist ein Python-Skript, das die beschriebenen Anforderungen
    erfüllt. Dieses Skript verwendet die 'pandas' Bibliothek, um alle Sheets
    in der Excel-Datei zu lesen, die Gesamtzahl der Straftaten für ganz Berlin
    zu aggregieren und die prozentuale Veränderung zum Vorjahr zu berechnen.
2
3 ### Voraussetzungen:
4 - Installiere die benötigten Bibliotheken, falls noch nicht geschehen:
5     '''bash
6     pip install pandas openpyxl
7     '''
8
9 ### Python-Skript:
10
11 '''python
12 import pandas as pd
13
14 # Pfad zur Excel-Datei
15 excel_file = 'Fallzahlen.xlsx'
16
17 # Lese alle Sheets in der Excel-Datei
18 # sheet_name=None lädt alle Sheets und gibt ein Dictionary zurück
19 sheets_dict = pd.read_excel(excel_file, sheet_name=None)
20
21 # Liste zur Speicherung der jährlichen Gesamtstraftaten
22 data = []
23
24 # Iteriere über alle Sheets
25 for sheet_name, df in sheets_dict.items():
26     # Überprüfe, ob 'LOR-Schlüssel' und 'Straftaten_insgesamt' in den Spalten
        vorhanden sind
27     if 'LOR-Schlüssel' in df.columns and 'Straftaten_insgesamt' in df.columns
        :
28         # Suche die Zeile, die die Gesamtzahl für Berlin enthält
29         # Annahme: 'LOR-Schlüssel' 999999 oder 'Berlin (PKS gesamt)' reprä
        sentiert die Gesamtdaten
30         total_row = df[df['LOR-Schlüssel'] == 999999]
31
32         # Falls nicht gefunden, versuche mit dem Namen
33         if total_row.empty:
34             total_row = df[df['LOR-Schlüssel'].astype(str).str.contains('

```



```

Berlin \((PKS gesamt\)`, regex=True)]

35
36     # Falls immer noch nicht gefunden, überspringe das Sheet
37     if total_row.empty:
38         print(f"Warnung: Gesamtdaten nicht in Sheet '{sheet_name}'
gefunden.")
39         continue
40
41     # Extrahiere das Jahr aus dem Sheet-Namen
42     # Annahme: Der Sheet-Name enthält das Jahr, z.B. "2020", "Jahr 2020",
etc.
43     # Hier wird versucht, eine vierstellige Zahl zu extrahieren
44     import re
45     match = re.search(r'\b(19|20)\d{2}\b', sheet_name)
46     if match:
47         year = int(match.group())
48     else:
49         print(f"Warnung: Jahr konnte nicht aus dem Sheet-Namen '{
sheet_name}' extrahiert werden. Überspringe dieses Sheet.")
50         continue
51
52     # Extrahiere die Gesamtzahl der Straftaten
53     total_crimes = total_row['Straftaten_insgesamt'].values[0]
54
55     # Füge die Daten zur Liste hinzu
56     data.append({'Jahr': year, 'Straftaten_insgesamt': total_crimes})
57     else:
58         print(f"Warnung: Erforderliche Spalten nicht in Sheet '{sheet_name}'
vorhanden.")
59
60 # Erstelle einen DataFrame aus den gesammelten Daten
61 df_total = pd.DataFrame(data)
62
63 # Sortiere den DataFrame nach dem Jahr
64 df_total = df_total.sort_values('Jahr').reset_index(drop=True)
65
66 # Berechne die prozentuale Veränderung zum Vorjahr
67 df_total['Prozentuale_Veraenderung_zum_Vorjahr (%)'] = df_total['
Straftaten_insgesamt'].pct_change() * 100
68
69 # Optional: Rundung der prozentualen Veränderung auf zwei Dezimalstellen
70 df_total['Prozentuale_Veraenderung_zum_Vorjahr (%)'] = df_total['
Prozentuale_Veraenderung_zum_Vorjahr (%)'].round(2)
71
72 # Setze das Jahr als Index (optional)
73 df_total.set_index('Jahr', inplace=True)
74
75 # Anzeige des Ergebnis-DataFrames
76 print(df_total)
77
78 # Optional: Speichere das Ergebnis in eine neue Excel-Datei
79 df_total.to_excel('Zeitreihe_Straftaten_Berlin.xlsx')

```

```

80 '''
81
82 ### Erklärung der Schritte:
83
84 1. **Importieren der Bibliotheken:**
85     - 'pandas' zum Datenmanipulieren.
86     - 're' zum Extrahieren des Jahres aus dem Sheet-Namen.
87
88 2. **Lesen aller Sheets:**
89     - 'pd.read_excel' mit 'sheet_name=None' lädt alle Sheets und speichert sie
90       in einem Dictionary.
91
92 3. **Datenextraktion:**
93     - Für jedes Sheet wird nach der Zeile gesucht, die die Gesamtzahl der
94       Straftaten für Berlin enthält ('LOR-Schlüssel' 999999 oder entsprechendem
95       Namen).
96     - Das Jahr wird aus dem Sheet-Namen extrahiert. Es wird angenommen, dass
97       der Sheet-Name das Jahr enthält (z.B. "2020", "Jahr 2020").
98
99 4. **Aggregation:**
100     - Die Gesamtzahl der Straftaten für jedes Jahr wird in einer Liste
101       gesammelt.
102
103 5. **Erstellung des DataFrames:**
104     - Die gesammelten Daten werden in einen DataFrame umgewandelt, nach Jahr
105       sortiert und die prozentuale Veränderung zum Vorjahr berechnet.
106
107 6. **Ausgabe:**
108     - Der resultierende DataFrame wird angezeigt und optional in eine neue
109       Excel-Datei gespeichert.
110
111 ### Hinweise:
112
113 - **Sheet-Namen:** Das Skript geht davon aus, dass die Sheet-Namen das Jahr
114   enthalten. Falls die Sheet-Namen ein anderes Format haben, muss die Regex-
115   Anpassung ('re.search') entsprechend geändert werden.
116
117 - **Fehlerbehandlung:** Das Skript gibt Warnungen aus, wenn bestimmte Sheets
118   nicht die erwarteten Daten enthalten. Dies hilft, Probleme bei der
119   Datenqualität zu identifizieren.
120
121 - **Flexibilität:** Falls die Gesamtdaten anders identifiziert werden müssen
122   (z.B. andere 'LOR-Schlüssel'), passe die Bedingung entsprechend an.
123
124 ### Beispiel-Ausgabe:
125
126 Angenommen, die Excel-Datei enthält Daten für die Jahre 2018 bis 2023, könnte
127 die Ausgabe wie folgt aussehen:
128
129 '''
130
131     Straftaten_insgesamt    Prozentuale_Veraenderung_zum_Vorjahr (%)
132 Jahr

```

```

119 2018                500000                NaN
120 2019                510000                2.00
121 2020                495000               -2.94
122 2021                530000                7.07
123 2022                550000                3.77
124 2023                540000               -1.82
125 '''
126
127 Diese Tabelle zeigt die Gesamtzahl der Straftaten pro Jahr sowie die
    prozentuale Veränderung im Vergleich zum Vorjahr.
128
129 ---
130
131 Ich hoffe, dieses Skript hilft dir bei der Analyse deiner Daten! Falls du
    weitere Anpassungen benötigst oder Fragen hast, stehe ich gerne zur Verfü-
    gung.

```

Listing 114: AI Response Testfall 4 Prompt 1 Ausführung 4

```

1 Hier ist ein Python-Skript, das Ihre Anforderungen erfüllt. Es liest alle
  Sheets aus der Excel-Datei 'Fallzahlen.xlsx', aggregiert die Gesamtzahl
  der Straftaten für ganz Berlin, berechnet die prozentuale Veränderung zum
  Vorjahr und erstellt eine Zeitreihe als Pandas DataFrame.
2
3 '''python
4 import pandas as pd
5
6 # Pfad zur Excel-Datei
7 excel_file = 'Fallzahlen.xlsx'
8
9 # Excel-Datei laden
10 xls = pd.ExcelFile(excel_file)
11
12 # Liste zur Speicherung der jährlichen Gesamtdaten
13 jahresdaten = []
14
15 # Annahme: Die Sheet-Namen entsprechen den Jahreszahlen (z.B. '2018', '2019',
    ...)
16 for sheet in xls.sheet_names:
17     # Daten des aktuellen Sheets lesen
18     df = pd.read_excel(xls, sheet_name=sheet)
19
20     # Sicherstellen, dass die 'LOR-Schlüssel' Spalte als String behandelt
    wird
21     df['LOR-Schlüssel'] = df['LOR-Schlüssel'].astype(str)
22
23     # Filtern nach der Zeile für ganz Berlin (angenommen '999999' ist der
    Schlüssel)
24     berlin_gesamt = df[df['LOR-Schlüssel'] == '999999']
25
26     if not berlin_gesamt.empty:
27         # Gesamtzahl der Straftaten für Berlin aus der entsprechenden Zeile
28         straftaten = berlin_gesamt['Straftaten_insgesamt'].values[0]

```

```

29     else:
30         # Wenn die 'Berlin (PKS gesamt)' Zeile nicht vorhanden ist, summiere
           alle Bezirke
31         straftaten = df['Straftaten_insgesamt'].sum()
32
33         # Füge die Daten zur Liste hinzu
34         jahresdaten.append({'Jahr': sheet, 'Straftaten_insgesamt': straftaten})
35
36 # Erstelle einen DataFrame aus den gesammelten Daten
37 df_gesamt = pd.DataFrame(jahresdaten)
38
39 # Versuche, die 'Jahr' Spalte in Integer zu konvertieren für die Sortierung
40 try:
41     df_gesamt['Jahr'] = df_gesamt['Jahr'].astype(int)
42     df_gesamt = df_gesamt.sort_values('Jahr')
43 except ValueError:
44     # Falls die Sheet-Namen nicht numerisch sind, sortiere alphabetisch
45     df_gesamt = df_gesamt.sort_values('Jahr')
46
47 # Setze das Jahr als Index
48 df_gesamt.set_index('Jahr', inplace=True)
49
50 # Berechne die prozentuale Veränderung zum Vorjahr
51 df_gesamt['% Veränderung Vorjahr'] = df_gesamt['Straftaten_insgesamt'].
           pct_change() * 100
52
53 # Runde die Prozentwerte auf zwei Dezimalstellen
54 df_gesamt['% Veränderung Vorjahr'] = df_gesamt['% Veränderung Vorjahr'].round
           (2)
55
56 # Anzeige des resultierenden DataFrames
57 print(df_gesamt)
58
59 # Optional: Speichern des DataFrames in eine neue Excel-Datei
60 df_gesamt.to_excel('Prozentuale_Veraenderung_Straftaten_Berlin.xlsx')
61 '''
62
63 ### Schritt-für-Schritt-Erklärung:
64
65 1. **Importieren der notwendigen Bibliothek:**
66     '''python
67     import pandas as pd
68     '''
69     Pandas wird für die Datenverarbeitung und Analyse verwendet.
70
71 2. **Laden der Excel-Datei:**
72     '''python
73     excel_file = 'Fallzahlen.xlsx'
74     xls = pd.ExcelFile(excel_file)
75     '''
76     Die Excel-Datei wird geöffnet, und alle Sheet-Namen werden erfasst.
77

```

```

78 3. **Iterieren durch alle Sheets:**
79     '''python
80     for sheet in xls.sheet_names:
81         df = pd.read_excel(xls, sheet_name=sheet)
82         ...
83     '''
84     Jedes Sheet wird einzeln gelesen und verarbeitet.
85
86 4. **Aggregieren der Straftaten für ganz Berlin:**
87     '''python
88     df['LOR-Schlüssel'] = df['LOR-Schlüssel'].astype(str)
89     berlin_gesamt = df[df['LOR-Schlüssel'] == '999999']
90
91     if not berlin_gesamt.empty:
92         straftaten = berlin_gesamt['Straftaten_insgesamt'].values[0]
93     else:
94         straftaten = df['Straftaten_insgesamt'].sum()
95     '''
96     Es wird nach der Zeile mit 'LOR-Schlüssel' '999999' gesucht, die die
97     Gesamtzahl der Straftaten für Berlin enthält. Falls diese Zeile nicht
98     existiert, werden die 'Straftaten_insgesamt' aller Bezirke summiert.
99
100 5. **Erstellen des DataFrames mit Jahresdaten:**
101     '''python
102     jahresdaten.append({'Jahr': sheet, 'Straftaten_insgesamt': straftaten})
103     df_gesamt = pd.DataFrame(jahresdaten)
104     '''
105
106 6. **Sortieren und Indexieren:**
107     '''python
108     try:
109         df_gesamt['Jahr'] = df_gesamt['Jahr'].astype(int)
110         df_gesamt = df_gesamt.sort_values('Jahr')
111     except ValueError:
112         df_gesamt = df_gesamt.sort_values('Jahr')
113
114     df_gesamt.set_index('Jahr', inplace=True)
115     '''
116
117 7. **Berechnung der prozentualen Veränderung zum Vorjahr:**
118     '''python
119     df_gesamt['% Veränderung Vorjahr'] = df_gesamt['Straftaten_insgesamt'].
120     pct_change() * 100
121     df_gesamt['% Veränderung Vorjahr'] = df_gesamt['% Veränderung Vorjahr'].
122     round(2)
123     '''
124
125 8. **Anzeige und optionales Speichern des Ergebnisses:**
126     '''python
127     print(df_gesamt)
128     df_gesamt.to_excel('Prozentuale_Veraenderung_Straftaten_Berlin.xlsx')
129     '''

```

```

126
127 ### Wichtige Hinweise:
128
129 - **Sheet-Namen:** Das Skript setzt voraus, dass die Sheet-Namen die
    Jahreszahlen sind (z.B. '2018', '2019'). Falls die Sheet-Namen ein anderes
    Format haben, müssen ggf. Anpassungen vorgenommen werden, insbesondere
    bei der Sortierung.
130
131 - **'Berlin (PKS gesamt)' Zeile:** Es wird angenommen, dass die Zeile mit '
    LOR-Schlüssel' '999999' die Gesamtzahlen für Berlin enthält. Sollte dies
    anders sein, passen Sie den Schlüssel entsprechend an.
132
133 - **Fehlende Daten:** Wenn bestimmte Sheets keine Gesamtzahlen für Berlin
    enthalten und die Summe der Bezirke nicht die korrekten Gesamtzahlen
    widerspiegelt, sollte eine alternative Methode zur Aggregation der Daten
    verwendet werden.
134
135 ### Voraussetzungen:
136
137 Stellen Sie sicher, dass die 'pandas' Bibliothek installiert ist. Falls nicht
    , können Sie sie mit folgendem Befehl installieren:
138
139 ```bash
140 pip install pandas
141 ```
142
143 ### Ausführen des Skripts:
144
145 1. Speichern Sie das Skript in einer Datei, z.B. 'analyse_fallzahlen.py'.
146 2. Stellen Sie sicher, dass sich die 'Fallzahlen.xlsx' im selben Verzeichnis
    wie das Skript befindet oder passen Sie den Pfad entsprechend an.
147 3. Führen Sie das Skript über die Kommandozeile aus:
148
149 ```bash
150 python analyse_fallzahlen.py
151 ```
152
153 Nach der Ausführung erhalten Sie eine Ausgabe des DataFrames mit den
    prozentualen Veränderungen der Straftaten für jedes Jahr und eine neue
    Excel-Datei 'Prozentuale_Veraenderung_Straftaten_Berlin.xlsx' mit den
    Ergebnissen.

```

Listing 115: AI Response Testfall 4 Prompt 1 Ausführung 5

```

1 Hier ist ein vollständiges Python-Skript, das die von dir gewünschte Analyse
  durchführt. Das Skript liest alle relevanten Sheets aus der Excel-Datei '
  Fallzahlen.xlsx', extrahiert die 'Straftaten_insgesamt'-Werte für "Berlin
  (PKS gesamt)" und berechnet die prozentuale Veränderung zum Vorjahr. Das
  Ergebnis wird als Pandas DataFrame ausgegeben.
2
3 ## Voraussetzungen
4
5 Stelle sicher, dass du die benötigten Bibliotheken installiert hast. Du

```

```

    kannst sie mit 'pip' installieren, falls sie noch nicht vorhanden sind:
6
7  '''bash
8  pip install pandas openpyxl
9  '''
10
11  ## Python-Skript
12
13  '''python
14  import pandas as pd
15
16  # Pfad zur Excel-Datei
17  excel_file = 'Fallzahlen.xlsx'
18
19  # Lade alle Sheet-Namen
20  xls = pd.ExcelFile(excel_file)
21  sheets = xls.sheet_names
22
23  # Extrahiere das Jahr aus den Sheet-Namen und sortiere die Sheets nach Jahr
24  sheet_years = []
25  for sheet in sheets:
26      try:
27          # Erwartetes Format: 'Fallzahlen_2014', 'Fallzahlen_2015', etc.
28          year = int(sheet.split('_')[-1])
29          sheet_years.append((year, sheet))
30      except ValueError:
31          print(f"Sheet-Name '{sheet}' enthält kein gültiges Jahr und wird ü
32          bersprungen.")
33          continue
34
35  # Sortiere die Sheets nach Jahr
36  sheet_years_sorted = sorted(sheet_years, key=lambda x: x[0])
37
38  # Liste zur Speicherung der Ergebnisse
39  data = []
40
41  # Iteriere über jedes Sheet und extrahiere die benötigten Daten
42  for year, sheet in sheet_years_sorted:
43      # Lese das aktuelle Sheet
44      df = pd.read_excel(excel_file, sheet_name=sheet)
45
46      # Stelle sicher, dass die benötigten Spalten vorhanden sind
47      if 'Bezirke' not in df.columns or 'Straftaten_insgesamt' not in df.
48      columns:
49          print(f"Sheet '{sheet}' fehlt die erforderliche Spalte. Überspringe
50          dieses Sheet.")
51          continue
52
53      # Filtere die Zeile für "Berlin (PKS gesamt)"
54      row = df[df['Bezirke'] == "Berlin (PKS gesamt)"]
55
56      if not row.empty:

```

```

54         # Extrahiere den Wert und stelle sicher, dass er numerisch ist
55         value = row['Straftaten_insgesamt'].values[0]
56         if pd.notnull(value):
57             data.append({'Year': year, 'Straftaten_insgesamt': value})
58         else:
59             print(f"Der Wert für 'Straftaten_insgesamt' im Jahr {year} ist
60             NaN.")
61         else:
62             print(f"'Berlin (PKS gesamt)' nicht in Sheet '{sheet}' gefunden.")
63
64 # Erstelle ein DataFrame aus den gesammelten Daten
65 df_data = pd.DataFrame(data)
66
67 # Sortiere das DataFrame nach Jahr, um sicherzustellen, dass die Reihenfolge
68 # stimmt
69 df_data = df_data.sort_values('Year').reset_index(drop=True)
70
71 # Berechne die prozentuale Veränderung zum Vorjahr
72 df_data['Prozentuale_Veraenderung'] = df_data['Straftaten_insgesamt'].
73     pct_change() * 100
74
75 # Optional: Runde die Prozentwerte auf zwei Dezimalstellen
76 df_data['Prozentuale_Veraenderung'] = df_data['Prozentuale_Veraenderung'].
77     round(2)
78
79 # Anzeige des Ergebnisses
80 print(df_data)
81
82 # Optional: Speichere das Ergebnis in eine neue Excel-Datei oder CSV
83 # df_data.to_excel('Prozentuale_Veraenderung_Fallzahlen.xlsx', index=False)
84 # df_data.to_csv('Prozentuale_Veraenderung_Fallzahlen.csv', index=False)
85
86
87
88
89
90
91
92
93
94
95
96
97

```

## Erklärung der einzelnen Schritte

1. **Importieren der Bibliotheken**:
  - 'pandas' wird für die Datenverarbeitung verwendet.
2. **Laden der Excel-Datei**:
  - Mit 'pd.ExcelFile' werden alle Sheet-Namen aus der Excel-Datei extrahiert.
3. **Extrahieren und Sortieren der Jahre**:
  - Die Sheet-Namen werden erwartet im Format 'Fallzahlen\_Jahr' (z.B. 'Fallzahlen\_2014').
  - Das Jahr wird extrahiert und die Sheets werden in aufsteigender Reihenfolge sortiert.
4. **Datenextraktion je Sheet**:
  - Für jedes sortierte Sheet wird das entsprechende DataFrame geladen.
  - Es wird überprüft, ob die notwendigen Spalten ('Bezirke' und 'Straftaten\_insgesamt') vorhanden sind.



```

98 - Die Zeile für "Berlin (PKS gesamt)" wird gefiltert und der Wert der '
    Straftaten_insgesamt' extrahiert.
99
100 5. **Erstellung des Ergebnis-DataFrames**:
101 - Die gesammelten Daten werden in ein neues DataFrame 'df_data' überführt.
102 - Es wird nach 'Year' sortiert, um die korrekte Reihenfolge
    sicherzustellen.
103
104 6. **Berechnung der prozentualen Veränderung**:
105 - Mit 'pct_change()' wird die prozentuale Veränderung der '
    Straftaten_insgesamt' zum Vorjahr berechnet.
106 - Die Ergebnisse werden in der neuen Spalte 'Prozentuale_Veraenderung'
    gespeichert und optional auf zwei Dezimalstellen gerundet.
107
108 7. **Ausgabe und Speicherung**:
109 - Das resultierende DataFrame wird ausgegeben.
110 - Optional kann das Ergebnis in eine neue Excel- oder CSV-Datei
    gespeichert werden, indem die entsprechenden Zeilen entkommentiert werden.
111
112 ## Beispielhafte Ausgabe
113
114 Angenommen, die extrahierten 'Straftaten_insgesamt'-Werte über die Jahre sind
    wie folgt:
115
116 | Year | Straftaten_insgesamt | Prozentuale_Veraenderung |
117 |-----|-----|-----|
118 | 2014 | 50000 | NaN |
119 | 2015 | 52000 | 4.00 |
120 | 2016 | 51000 | -1.92 |
121 | 2017 | 53000 | 3.92 |
122 | ... | ... | ... |
123
124 'NaN' erscheint für das erste Jahr, da es keinen Vorjahreswert gibt.
125
126 ## Hinweise
127
128 - **Fehlende Daten**: Das Skript gibt Warnungen aus, wenn bestimmte Sheets
    nicht den erwarteten Aufbau haben oder wenn "Berlin (PKS gesamt)" nicht
    gefunden wird.
129 - **Anpassungen**: Falls die Sheet-Namen ein anderes Format haben oder zusätzliche
    Bereinigungen erforderlich sind, muss das Skript entsprechend
    angepasst werden.
130 - **Speicherung**: Die letzten beiden Zeilen im Skript sind auskommentiert.
    Entferne das Kommentarzeichen ('#'), um die Ergebnisse in eine Datei zu
    speichern.
131
132 Ich hoffe, dieses Skript hilft dir bei deiner Analyse! Wenn du weitere
    Anpassungen benötigst oder Fragen hast, stehe ich gerne zur Verfügung.

```

Listing 116: AI Response Testfall 4 Prompt 2 Ausführung 1

```

1 Hier ist ein vollständiges Python-Skript, das die gewünschten Anforderungen
    erfüllt. Dieses Skript liest die Excel-Datei 'Fallzahlen.xlsx', extrahiert

```

```

    die relevanten Daten aus jedem Sheet, berechnet die prozentuale Veränderung der Spalte 'Straftaten_insgesamt' zum Vorjahr für den Eintrag "Berlin (PKS gesamt)" und erstellt schließlich eine Zeitreihe als Pandas DataFrame.
2
3 Stellen Sie sicher, dass Sie die benötigten Bibliotheken installiert haben. Sie können dies mit 'pip' tun:
4
5 '''bash
6 pip install pandas openpyxl
7 '''
8
9 Hier ist das Python-Skript:
10
11 '''python
12 import pandas as pd
13
14 def compute_percentage_change(excel_file, target_row, target_column, sheet_prefix):
15     """
16     Liest eine Excel-Datei mit mehreren Sheets, extrahiert die Straftaten_insgesamt für einen bestimmten Eintrag und berechnet die prozentuale Veränderung zum Vorjahr.
17
18     Parameters:
19     - excel_file: str, Pfad zur Excel-Datei.
20     - target_row: str, Der Eintrag in der 'Bezirke' Spalte, z.B. "Berlin (PKS gesamt)".
21     - target_column: str, Name der Spalte, deren prozentuale Veränderung berechnet werden soll, z.B. "Straftaten_insgesamt".
22     - sheet_prefix: str, Präfix der Sheet-Namen, z.B. "Fallzahlen_" für Sheets wie "Fallzahlen_2014".
23
24     Returns:
25     - pandas.DataFrame mit den Jahren und der prozentualen Veränderung.
26     """
27
28     try:
29         # Alle Sheets einlesen
30         sheets_dict = pd.read_excel(excel_file, sheet_name=None, engine='openpyxl')
31     except FileNotFoundError:
32         print(f"Die Datei {excel_file} wurde nicht gefunden.")
33         return None
34     except Exception as e:
35         print(f"Ein Fehler ist aufgetreten beim Einlesen der Datei: {e}")
36         return None
37
38     data = []
39
40     for sheet_name, df in sheets_dict.items():
41         # Überprüfen, ob der Sheet-Name dem erwarteten Muster entspricht

```

```

42         if not sheet_name.startswith(sheet_prefix):
43             print(f"Überspringe Sheet '{sheet_name}', da es nicht mit '{
sheet_prefix}' beginnt.")
44             continue
45
46         # Extrahiere das Jahr aus dem Sheet-Namen
47         try:
48             year_str = sheet_name.replace(sheet_prefix, "")
49             year = int(year_str)
50         except ValueError:
51             print(f"Konnte das Jahr aus dem Sheet-Namen '{sheet_name}' nicht
extrahieren.")
52             continue
53
54         # Suche die Zeile mit dem gewünschten Eintrag
55         row = df[df['Bezirke'] == target_row]
56
57         if row.empty:
58             print(f"Der Eintrag '{target_row}' wurde in Sheet '{sheet_name}'
nicht gefunden.")
59             continue
60
61         # Extrahiere den Wert der Zielspalte
62         try:
63             value = row.iloc[0][target_column]
64             data.append({'Year': year, target_column: value})
65         except KeyError:
66             print(f"Die Spalte '{target_column}' wurde in Sheet '{sheet_name
}' nicht gefunden.")
67             continue
68
69         if not data:
70             print("Keine Daten gefunden, um die prozentuale Veränderung zu
berechnen.")
71             return None
72
73         # Erstelle einen DataFrame aus den gesammelten Daten
74         df_data = pd.DataFrame(data)
75
76         # Sortiere die Daten nach Jahr
77         df_data = df_data.sort_values('Year').reset_index(drop=True)
78
79         # Berechne die prozentuale Veränderung zum Vorjahr
80         df_data['Percentage_Change'] = df_data[target_column].pct_change() * 100
81
82         return df_data[['Year', 'Percentage_Change']]
83
84 if __name__ == "__main__":
85     # Parameter festlegen
86     excel_file = 'Fallzahlen.xlsx'
87     target_row = 'Berlin (PKS gesamt)'
88     target_column = 'Straftaten_insgesamt'

```

```

89     sheet_prefix = 'Fallzahlen_'
90
91     # Funktion aufrufen
92     percentage_change_df = compute_percentage_change(excel_file, target_row,
93                                                       target_column, sheet_prefix)
94
95     if percentage_change_df is not None:
96         print("Prozentuale Veränderung der Straftaten_insgesamt zum Vorjahr:"
97               )
98         print(percentage_change_df)
99
100         # Optional: DataFrame speichern
101         # percentage_change_df.to_csv('prozentuale_veraenderung.csv', index=
102         False)
103
104     """
105
106     ### Erklärung des Skripts
107
108     1. **Import der Bibliotheken**:
109         - 'pandas' wird für die Datenverarbeitung verwendet.
110
111     2. **Funktion 'compute_percentage_change'**:
112         - **Parameter**:
113             - 'excel_file': Pfad zur Excel-Datei.
114             - 'target_row': Der spezifische Eintrag in der Spalte 'Bezirke', z.B. "
115               Berlin (PKS gesamt)".
116             - 'target_column': Die Spalte, für die die prozentuale Veränderung
117               berechnet werden soll, z.B. "Straftaten_insgesamt".
118             - 'sheet_prefix': Das Präfix der Sheet-Namen, z.B. "Fallzahlen_".
119         - **Prozess**:
120             - Öffnet die Excel-Datei und liest alle Sheets ein.
121             - Iteriert durch jedes Sheet, extrahiert das Jahr aus dem Sheet-Namen.
122             - Sucht die Zeile, die dem 'target_row' entspricht.
123             - Extrahiert den Wert der 'target_column' aus dieser Zeile.
124             - Speichert das Jahr und den extrahierten Wert in einer Liste.
125             - Nachdem alle Daten gesammelt wurden, wird ein DataFrame erstellt, nach
126               Jahr sortiert und die prozentuale Veränderung berechnet.
127
128     3. **Hauptblock ('if __name__ == "__main__":')**:
129         - Setzt die erforderlichen Parameter.
130         - Ruft die Funktion 'compute_percentage_change' auf.
131         - Wenn Daten gefunden werden, werden diese ausgegeben. Optional kann der
132           resultierende DataFrame auch in eine CSV-Datei gespeichert werden (
133           auskommentiert).
134
135     ### Beispielausgabe
136
137     Angenommen, die Excel-Datei enthält Daten von 2014 bis 2023, könnte die
138       Ausgabe wie folgt aussehen:
139
140     """
141     Prozentuale Veränderung der Straftaten_insgesamt zum Vorjahr:

```

```

132     Year    Percentage_Change
133 0    2014                NaN
134 1    2015             5.123456
135 2    2016            -2.345678
136 3    2017             3.210987
137 4    2018            -1.112233
138 5    2019             4.567890
139 6    2020            -3.210987
140 7    2021             2.345678
141 8    2022             0.987654
142 9    2023             1.234567
143 '''
144
145 *Hinweis*: Der Wert für das Jahr 2014 ist 'NaN', da es keinen Vorjahreswert
    gibt, mit dem die Veränderung berechnet werden könnte.
146
147 ### Anpassungen und Erweiterungen
148
149 - **Fehlerbehandlung**: Das Skript enthält grundlegende Fehlerbehandlungen,
    um sicherzustellen, dass fehlende Dateien, nicht gefundene Sheets oder
    fehlende Spalten ordnungsgemäß behandelt werden.
150 - **Flexibilität**: Durch die Verwendung von Parametern können Sie das Skript
    leicht anpassen, um andere Zeilen oder Spalten zu analysieren.
151 - **Speichern der Ergebnisse**: Es besteht die Möglichkeit, die Ergebnisse in
    eine CSV-Datei zu speichern, indem Sie die entsprechende Zeile
    entkommentieren.
152
153 ### Ausführung des Skripts
154
155 Speichern Sie das Skript in einer Datei, z.B. 'analyse_fallzahlen.py', und fü
    hren Sie es in Ihrer Kommandozeile aus:
156
157 '''bash
158 python analyse_fallzahlen.py
159 '''
160
161 Stellen Sie sicher, dass sich 'Fallzahlen.xlsx' im gleichen Verzeichnis wie
    das Skript befindet oder geben Sie den vollständigen Pfad zur Excel-Datei
    an.

```

Listing 117: AI Response Testfall 4 Prompt 2 Ausführung 2

```

1 Natürlich, hier ist ein Python-Skript, das die von dir beschriebenen
    Anforderungen erfüllt. Dieses Skript verwendet die 'pandas' Bibliothek, um
    die Excel-Datei zu lesen, die relevanten Daten zu extrahieren und die
    prozentuale Veränderung der Straftaten insgesamt Klasse pro Jahr zu
    berechnen.
2
3 ### Voraussetzungen
4
5 Stelle sicher, dass du 'pandas' und 'openpyxl' installiert hast. Falls nicht,
    kannst du sie mit 'pip' installieren:
6

```

```

7  '''bash
8  pip install pandas openpyxl
9  '''
10
11  ### Python-Skript
12
13  '''python
14  import pandas as pd
15
16  # Pfad zur Excel-Datei
17  excel_file = 'Fallzahlen.xlsx'
18
19  # Lese alle Sheetnamen aus der Excel-Datei
20  xls = pd.ExcelFile(excel_file)
21  sheet_names = xls.sheet_names
22
23  # Initialisiere ein Dictionary, um die Daten zu speichern
24  data = {}
25
26  # Iteriere durch alle Sheets
27  for sheet in sheet_names:
28      try:
29          # Extrahiere das Jahr aus dem Sheetnamen, z.B. 'Fallzahlen_2014' ->
2014
30          year = int(sheet.split('_')[-1])
31      except ValueError:
32          print(f"Sheet '{sheet}' folgt nicht dem erwarteten Namensformat. Ü
berspringe...")
33          continue
34
35      # Lese das aktuelle Sheet
36      df = pd.read_excel(xls, sheet_name=sheet)
37
38      # Filtere die Zeile für 'Berlin (PKS gesamt)'
39      berlin_data = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
40
41      if berlin_data.empty:
42          print(f"Keine Daten für 'Berlin (PKS gesamt)' im Sheet '{sheet}'
gefunden.")
43          continue
44
45      # Extrahiere den Wert der Spalte 'Straftaten_insgesamt'
46      total_crimes = berlin_data['Straftaten_insgesamt'].values[0]
47
48      # Speichere die Daten im Dictionary
49      data[year] = total_crimes
50
51  # Erstelle einen DataFrame aus dem Dictionary
52  df_crimes = pd.DataFrame(list(data.items()), columns=['Jahr', '
Straftaten_insgesamt'])
53
54  # Sortiere den DataFrame nach Jahr

```

```

55 df_crimes = df_crimes.sort_values('Jahr').reset_index(drop=True)
56
57 # Berechne die prozentuale Veränderung zum Vorjahr
58 df_crimes['Prozentuale_Veränderung (%)'] = df_crimes['Straftaten_insgesamt'].
    pct_change() * 100
59
60 # Optional: Runde die prozentuale Veränderung auf zwei Dezimalstellen
61 df_crimes['Prozentuale_Veränderung (%)'] = df_crimes['Prozentuale_Veränderung
    (%)'].round(2)
62
63 # Zeige den Ergebnis-DataFrame
64 print(df_crimes)
65
66 # Optional: Speichere den DataFrame in eine neue Excel-Datei
67 df_crimes.to_excel('Straftaten_Veränderung_Zeitreihe.xlsx', index=False)
68 '''
69
70 ### Erklärung des Skripts
71
72 1. **Importieren der Bibliotheken:**
73     - 'pandas' wird für die Datenmanipulation und -analyse verwendet.
74
75 2. **Excel-Datei laden:**
76     - Das Skript lädt die Excel-Datei 'Fallzahlen.xlsx' und extrahiert alle
        Sheetnamen.
77
78 3. **Daten extrahieren:**
79     - Für jedes Sheet wird das Jahr aus dem Sheetnamen extrahiert.
80     - Das Skript sucht nach der Zeile, wo die Spalte 'Bezirke' den Wert '
        Berlin (PKS gesamt)' hat.
81     - Der Wert aus der Spalte 'Straftaten_insgesamt' wird extrahiert und im
        Dictionary 'data' gespeichert.
82
83 4. **DataFrame erstellen:**
84     - Das Dictionary wird in einen 'pandas' DataFrame umgewandelt.
85     - Der DataFrame wird nach Jahr sortiert.
86
87 5. **Prozentuale Veränderung berechnen:**
88     - Die prozentuale Veränderung der Straftaten insgesamten Zahl zum Vorjahr
        wird berechnet und als neue Spalte hinzugefügt.
89
90 6. **Ergebnis anzeigen und speichern:**
91     - Der finale DataFrame wird angezeigt.
92     - Optional kann der DataFrame in eine neue Excel-Datei 'Straftaten_Verä
        nderung_Zeitreihe.xlsx' gespeichert werden.
93
94 ### Beispielausgabe
95
96 Nach dem Ausführen des Skripts erhältst du einen DataFrame ähnlich dem
    folgenden:
97
98 '''

```

```

99      Jahr  Straftaten_insgesamt  Prozentuale_Veränderung (%)
100 0   2014                    500000                      NaN
101 1   2015                    510000                      2.00
102 2   2016                    495000                     -2.94
103 3   2017                    530000                      7.07
104 4   2018                    550000                      3.77
105 ...
106 '''
107
108 - **Jahr:** Das Jahr der Daten.
109 - **Straftaten_insgesamt:** Gesamtzahl der Straftaten in Berlin.
110 - **Prozentuale_Veränderung (%):** Prozentuale Veränderung gegenüber dem
    Vorjahr.
111
112 ### Hinweise
113
114 - **Fehlerbehandlung:** Das Skript enthält einfache Fehlerbehandlungen, um
    Sheets zu überspringen, die nicht dem erwarteten Namensformat folgen oder
    in denen keine Daten für 'Berlin (PKS gesamt)' gefunden werden.
115
116 - **Anpassungen:** Stelle sicher, dass die Spaltennamen in der Excel-Datei
    exakt mit den im Skript verwendeten Namen übereinstimmen ('Bezirke' und
    'Straftaten_insgesamt'). Falls sie abweichen, passe die Namen im Skript
    entsprechend an.
117
118 - **Erweiterungen:** Du kannst das Skript erweitern, um zusätzliche Analysen
    durchzuführen oder Visualisierungen zu erstellen, zum Beispiel mit '
    matplotlib' oder 'seaborn'.
119
120 Ich hoffe, dieses Skript hilft dir bei deiner Analyse! Falls du weitere
    Fragen oder Anpassungswünsche hast, lass es mich wissen.

```

Listing 118: AI Response Testfall 4 Prompt 2 Ausführung 3

```

1 Hier ist ein vollständiges Python-Skript, das die gewünschten Analysen durchf
  ührt. Das Skript verwendet die 'pandas'-Bibliothek, um die Excel-Datei zu
  lesen, die relevanten Daten aus jedem Sheet zu extrahieren, die
  prozentuale Veränderung der Straftaten zum Vorjahr zu berechnen und das
  Ergebnis als Pandas DataFrame darzustellen.
2
3 Stellen Sie sicher, dass Sie die 'pandas'-Bibliothek installiert haben. Falls
  nicht, können Sie sie mit 'pip install pandas openpyxl' installieren, da
  'openpyxl' für das Lesen von Excel-Dateien benötigt wird.
4
5 '''python
6 import pandas as pd
7
8 def analyze_fallzahlen(file_path, target_district="Berlin (PKS gesamt)",
9                        column_name="Straftaten_insgesamt"):
10     """
11     Analysiert die Excel-Datei 'Fallzahlen.xlsx' und erstellt eine Zeitreihe
12     mit
    der prozentualen Veränderung der Straftaten zum jeweiligen Vorjahr.

```



```

13
14 Parameters:
15 - file_path: Pfad zur Excel-Datei
16 - target_district: Name des Bezirks, der analysiert werden soll
17 - column_name: Name der Spalte, deren Veränderung berechnet werden soll
18
19 Returns:
20 - pandas DataFrame mit den Jahreszahlen und der prozentualen Veränderung
21 ""
22
23 # Lade die Excel-Datei
24 try:
25     xls = pd.ExcelFile(file_path)
26 except FileNotFoundError:
27     print(f"Die Datei {file_path} wurde nicht gefunden.")
28     return
29 except Exception as e:
30     print(f"Ein Fehler ist aufgetreten beim Laden der Datei: {e}")
31     return
32
33 # Initialisiere ein Dictionary zur Speicherung der Ergebnisse
34 data = {}
35
36 # Iteriere über alle Sheets in der Excel-Datei
37 for sheet in xls.sheet_names:
38     # Extrahiere das Jahr aus dem Sheet-Namen
39     try:
40         year = int(sheet.split('_')[-1])
41     except ValueError:
42         print(f"Das Sheet '{sheet}' folgt nicht dem erwarteten
43 Namensformat. Überspringe dieses Sheet.")
44         continue
45
46     # Lese das aktuelle Sheet in einen DataFrame
47     try:
48         df = pd.read_excel(xls, sheet_name=sheet)
49     except Exception as e:
50         print(f"Ein Fehler ist aufgetreten beim Lesen des Sheets '{sheet}': {e}")
51         continue
52
53     # Finde die Zeile für den Zielbezirk
54     row = df[df['Bezirke'] == target_district]
55
56     if row.empty:
57         print(f"Der Bezirk '{target_district}' wurde im Sheet '{sheet}'
58 nicht gefunden. Überspringe dieses Sheet.")
59         continue
60
61     # Extrahiere den Wert der gewünschten Spalte
62     try:
63         value = row[column_name].values[0]

```

```

62         # Stelle sicher, dass der Wert numerisch ist
63         value = float(value)
64     except (IndexError, ValueError) as e:
65         print(f"Fehler beim Extrahieren des Wertes in Sheet '{sheet}': {e}
        })
66         continue
67
68     # Speichere das Jahr und den Wert im Dictionary
69     data[year] = value
70
71     # Erstelle einen DataFrame aus dem Dictionary
72     df_data = pd.DataFrame(list(data.items()), columns=['Year', column_name])
73
74     # Sortiere den DataFrame nach dem Jahr aufsteigend
75     df_data.sort_values('Year', inplace=True)
76     df_data.reset_index(drop=True, inplace=True)
77
78     # Berechne die prozentuale Veränderung zum Vorjahr
79     df_data['Percent Change'] = df_data[column_name].pct_change() * 100
80
81     # Optional: Runde die Prozentänderungen auf zwei Dezimalstellen
82     df_data['Percent Change'] = df_data['Percent Change'].round(2)
83
84     # Setze das Jahr als Index (optional)
85     # df_data.set_index('Year', inplace=True)
86
87     return df_data
88
89 if __name__ == "__main__":
90     # Pfad zur Excel-Datei
91     excel_file = 'Fallzahlen.xlsx'
92
93     # Führe die Analyse durch
94     result_df = analyze_fallzahlen(excel_file)
95
96     if result_df is not None:
97         # Zeige das Ergebnis an
98         print(result_df)
99
100        # Optional: Speichere das Ergebnis in einer neuen Excel- oder CSV-
        Datei
101        # result_df.to_excel('Straftaten_Prozentuale_Veraenderung.xlsx',
        index=False)
102        # oder
103        # result_df.to_csv('Straftaten_Prozentuale_Veraenderung.csv', index=
        False)
104    '''
105
106    ### Erklärung des Skripts:
107
108    1. **Importieren der Bibliothek:**
109    - 'pandas' wird für die Datenmanipulation und -analyse verwendet.

```

```

110
111 2. **Funktion 'analyze_fallzahlen':**
112   - **Parameter:**
113     - 'file_path': Der Pfad zur Excel-Datei.
114     - 'target_district': Der Bezirk, der analysiert werden soll (Standard: "
115       Berlin (PKS gesamt)").
116     - 'column_name': Die Spalte, deren Veränderung berechnet werden soll (
117       Standard: "Straftaten_insgesamt").
118   - **Schritte:**
119     - Lädt die Excel-Datei und iteriert über alle Sheets.
120     - Extrahiert das Jahr aus dem Sheet-Namen.
121     - Liest das Sheet in einen DataFrame und sucht die Zeile für den
122       Zielbezirk.
123     - Extrahiert den Wert der gewünschten Spalte und speichert ihn im
124       Dictionary 'data'.
125     - Nach der Iteration erstellt der DataFrame 'df_data' aus dem Dictionary
126       , sortiert nach Jahr.
127     - Berechnet die prozentuale Veränderung zum Vorjahr und fügt sie als
128       neue Spalte hinzu.
129
130 3. **Hauptblock ('if __name__ == "__main__":'):**
131   - Definiert den Pfad zur Excel-Datei.
132   - Ruft die Funktion 'analyze_fallzahlen' auf und speichert das Ergebnis in
133     'result_df'.
134   - Gibt das Ergebnis aus.
135   - Optional kann das Ergebnis in eine neue Excel- oder CSV-Datei
136     gespeichert werden (auskommentiert).
137
138 ### Beispielausgabe:
139
140 Angenommen, die Excel-Datei enthält Daten von 2014 bis 2023, könnte die
141 Ausgabe wie folgt aussehen:
142
143 '''
144
145     Year  Straftaten_insgesamt  Percent Change
146 0   2014                500000             NaN
147 1   2015                510000             2.00
148 2   2016                495000            -2.94
149 3   2017                530000             7.07
150 4   2018                550000             3.77
151 5   2019                540000            -1.82
152 6   2020                560000             3.70
153 7   2021                580000             3.57
154 8   2022                570000            -1.72
155 9   2023                590000             3.51
156
157 '''
158
159 **Hinweis:**
160   - Der erste Jahrseintrag hat keine prozentuale Veränderung zum Vorjahr und
161     zeigt daher 'NaN'.
162   - Stellen Sie sicher, dass alle Sheet-Namen dem Format 'Fallzahlen_Jahr'
163     entsprechen, z.B. 'Fallzahlen_2014', 'Fallzahlen_2015' usw.

```

151 - Falls die Struktur der Excel-Datei abweicht oder zusätzliche Fehlerbehandlungen benötigt werden, kann das Skript entsprechend angepasst werden.

Listing 119: AI Response Testfall 4 Prompt 2 Ausführung 4

```
1 Hier ist ein Python-Skript, das die Anforderungen erfüllt. Es liest die Excel
  -Datei 'Fallzahlen.xlsx', analysiert alle relevanten Sheets, extrahiert
  die 'Straftaten_insgesamt'-Werte für 'Berlin (PKS gesamt)' und berechnet
  die prozentuale Veränderung zum Vorjahr. Das Ergebnis wird als Pandas
  DataFrame ausgegeben.
2
3 ### Voraussetzungen
4
5 Stelle sicher, dass du die benötigten Bibliotheken installiert hast. Du
  kannst sie mit 'pip' installieren, falls sie noch nicht vorhanden sind:
6
7 ```bash
8 pip install pandas openpyxl
9 ```
10
11 ### Python-Skript
12
13 ```python
14 import pandas as pd
15 import re
16
17 def extract_year(sheet_name):
18     """
19     Extrahiert das Jahr aus dem Sheet-Namen.
20     Erwartet Format: 'Fallzahlen_YYYY'
21     """
22     match = re.search(r'Fallzahlen_(\d{4})', sheet_name)
23     if match:
24         return int(match.group(1))
25     else:
26         return None
27
28 def get_straftaten_insgesamt(excel_file, sheet_name, target_district):
29     """
30     Liest ein Sheet und extrahiert den Wert von 'Straftaten_insgesamt' für
31     den angegebenen Bezirk.
32     """
33     df = pd.read_excel(excel_file, sheet_name=sheet_name)
34
35     # Annahme: Die Bezeichnung des Bezirks ist in der Spalte 'Bezirke'
36     row = df[df['Bezirke'] == target_district]
37
38     if not row.empty:
39         return row.iloc[0]['Straftaten_insgesamt']
40     else:
41         print(f"Warnung: Bezirk '{target_district}' nicht in Sheet '{sheet_name}' gefunden.")
```

```

41         return None
42
43 def main():
44     excel_file = 'Fallzahlen.xlsx'
45     target_district = 'Berlin (PKS gesamt)'
46
47     # Lade alle Sheet-Namen
48     xls = pd.ExcelFile(excel_file)
49     sheet_names = xls.sheet_names
50
51     # Filtere die relevanten Sheets und extrahiere die Jahre
52     pattern = re.compile(r'^Fallzahlen_(\d{4})$')
53     sheets_with_year = []
54     for sheet in sheet_names:
55         match = pattern.match(sheet)
56         if match:
57             year = int(match.group(1))
58             sheets_with_year.append((year, sheet))
59
60     if not sheets_with_year:
61         print("Keine Sheets im erwarteten Format 'Fallzahlen_YYYY' gefunden."
62 )
63         return
64
65     # Sortiere die Sheets nach Jahr
66     sheets_with_year.sort(key=lambda x: x[0])
67
68     # Extrahiere die 'Straftaten_insgesamt' Werte
69     data = {}
70     for year, sheet in sheets_with_year:
71         value = get_straftaten_insgesamt(excel_file, sheet, target_district)
72         if value is not None:
73             data[year] = value
74
75     # Erstelle ein DataFrame
76     df = pd.DataFrame(list(data.items()), columns=['Jahr', 'Straftaten_insgesamt'])
77     df.sort_values('Jahr', inplace=True)
78     df.reset_index(drop=True, inplace=True)
79
80     # Berechne die prozentuale Veränderung zum Vorjahr
81     df['Prozentuale_Veraenderung'] = df['Straftaten_insgesamt'].pct_change()
82     * 100
83
84     # Optional: Runde die Prozentwerte auf zwei Dezimalstellen
85     df['Prozentuale_Veraenderung'] = df['Prozentuale_Veraenderung'].round(2)
86
87     print(df)
88
89 if __name__ == "__main__":
90     main()
91

```

```

90
91 ### Erklärung des Skripts
92
93 1. **Importieren der Bibliotheken**:
94     - 'pandas' für die Datenverarbeitung.
95     - 're' für reguläre Ausdrücke zur Verarbeitung der Sheet-Namen.
96
97 2. **Funktionen**:
98     - 'extract_year': Extrahiert das Jahr aus dem Sheet-Namen, wenn es dem
99       Muster 'Fallzahlen_YYYY' entspricht.
100
101     - 'get_straftaten_insgesamt': Liest ein bestimmtes Sheet und extrahiert
102       den Wert der Spalte 'Straftaten_insgesamt' für den angegebenen Bezirk ('
103       Berlin (PKS gesamt)').
104
105 3. **Hauptfunktion ('main')**:
106     - Lädt die Excel-Datei und alle Sheet-Namen.
107     - Filtert die Sheets, die dem Muster 'Fallzahlen_YYYY' entsprechen, und
108       extrahiert die Jahre.
109     - Sortiert die Sheets chronologisch nach Jahr.
110     - Extrahiert die 'Straftaten_insgesamt'-Werte für 'Berlin (PKS gesamt)'
111       aus jedem Sheet.
112     - Erstellt ein Pandas DataFrame mit den Jahren und den entsprechenden
113       Straftatenzahlen.
114     - Berechnet die prozentuale Veränderung der Straftaten zum Vorjahr.
115     - Gibt das resultierende DataFrame aus.
116
117 ### Beispielhafte Ausgabe
118
119 Nach dem Ausführen des Skripts könnte die Ausgabe ungefähr wie folgt aussehen
120 (die tatsächlichen Werte hängen von den Daten in deinen Sheets ab):
121
122 '''
123
124     Jahr  Straftaten_insgesamt  Prozentuale_Veraenderung
125 0   2014                500000                NaN
126 1   2015                510000                2.00
127 2   2016                495000               -2.94
128 3   2017                520000                4.04
129 4   2018                530000                1.92
130 5   2019                515000               -2.83
131 6   2020                525000                1.95
132 7   2021                535000                1.90
133 8   2022                550000                2.81
134 9   2023                560000                1.82
135 '''
136
137 ### Hinweise
138
139 - **Fehlende Daten**: Falls ein bestimmtes Jahr oder ein Sheet nicht den benö
140   tigten Bezirk enthält, gibt das Skript eine Warnung aus und überspringt
141   diesen Eintrag.
142
143 - **Anpassungen**: Je nach tatsächlicher Struktur deiner Excel-Datei (z.B.
144   andere Spaltennamen oder zusätzliche Filter) könnte es notwendig sein, das

```

```

Skript entsprechend anzupassen.
132 - **Output-Format**: Das Skript druckt das resultierende DataFrame in der
    Konsole aus. Du kannst es jedoch leicht modifizieren, um die Daten in eine
    CSV-Datei zu speichern oder weiter zu verarbeiten.
133
134 ### Speichern des DataFrames in eine CSV-Datei (optional)
135
136 Wenn du das Ergebnis in eine CSV-Datei speichern möchtest, kannst du
    folgendes hinzufügen, bevor du das DataFrame ausgibst:
137
138 '''python
139     # Speichere das DataFrame in eine CSV-Datei
140     df.to_csv('Prozentuale_Veraenderung_Straftaten.csv', index=False)
141     print("Das Ergebnis wurde in 'Prozentuale_Veraenderung_Straftaten.csv'
    gespeichert.")
142 '''
143
144 Füge diese Zeilen direkt nach der Berechnung der prozentualen Veränderung ('
    df['Prozentuale_Veraenderung'] = ...') ein.
145
146 Ich hoffe, dieses Skript hilft dir weiter! Falls du weitere Anpassungen benö-
    tigtst oder Fragen hast, lass es mich wissen.

```

Listing 120: AI Response Testfall 4 Prompt 2 Ausführung 5

```

1 Gerne erstelle ich dir ein Python-Skript, das die beschriebenen Schritte
    durchführt. Wir verwenden dabei die Bibliotheken 'pandas' und 'openpyxl'.
    Stelle sicher, dass du diese Bibliotheken installiert hast. Du kannst sie
    mit 'pip' installieren, falls noch nicht geschehen:
2
3 '''bash
4 pip install pandas openpyxl
5 '''
6
7 Hier ist das vollständige Skript:
8
9 '''python
10 import pandas as pd
11
12 # Schritt 1: Alle Sheets der Excel-Datei einlesen
13 excel_datei = 'Fallzahlen.xlsx'
14 xls = pd.ExcelFile(excel_datei)
15
16 # Dictionary zum Speichern der DataFrames
17 dataframes = {}
18
19 # Extrahiere die Jahreszahlen aus den Sheet-Namen und sortiere sie
20 sheet_jahre = []
21 for sheet in xls.sheet_names:
22     try:
23         # Annahme: Sheetnamen sind im Format 'Fallzahlen_Jahr', z.B. '
    Fallzahlen_2014'
24         jahr = int(sheet.split('_')[-1])

```

```

25     sheet_jahre.append((jahr, sheet))
26 except ValueError:
27     print(f"Sheet '{sheet}' hat kein gültiges Jahresformat und wird ü
    bersprungen.")
28
29 # Sortiere die Sheets nach Jahr
30 sheet_jahre.sort()
31
32 # Lade die DataFrames und speichere sie im Dictionary
33 for jahr, sheet in sheet_jahre:
34     df = pd.read_excel(xls, sheet_name=sheet)
35     dataframes[jahr] = df
36
37 # Schritt 2: Extrahiere 'Straftaten_insgesamt' für 'Berlin (PKS gesamt)'
38 straftaten_gesamt = {}
39
40 for jahr, df in dataframes.items():
41     # Filtern der Zeile 'Berlin (PKS gesamt)'
42     berlin_row = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
43     if not berlin_row.empty:
44         wert = berlin_row.iloc[0]['Straftaten_insgesamt']
45         straftaten_gesamt[jahr] = wert
46     else:
47         print(f"Die Zeile 'Berlin (PKS gesamt)' wurde im Jahr {jahr} nicht
    gefunden.")
48
49 # Schritt 3: Berechne die prozentuale Veränderung zum Vorjahr
50 jahre = sorted(straftaten_gesamt.keys())
51 prozentuale_veraenderung = {}
52
53 for i in range(1, len(jahre)):
54     aktuelles_jahr = jahre[i]
55     vorheriges_jahr = jahre[i-1]
56     wert_aktuell = straftaten_gesamt[aktuelles_jahr]
57     wert_vorher = straftaten_gesamt[vorheriges_jahr]
58     veraenderung = ((wert_aktuell - wert_vorher) / wert_vorher) * 100
59     prozentuale_veraenderung[aktuelles_jahr] = veraenderung
60
61 # Schritt 4: Ergebnisse in einem neuen DataFrame speichern
62 ergebnisse = pd.DataFrame({
63     'Jahr': list(prozentuale_veraenderung.keys()),
64     'Prozentuale Veränderung (%)': list(prozentuale_veraenderung.values())
65 })
66
67 # Optional: Prozentuale Veränderung mit zwei Dezimalstellen formatieren
68 ergebnisse['Prozentuale Veränderung (%)'] = ergebnisse['Prozentuale Verä
    nderung (%)'].round(2)
69
70 # Ergebnisse anzeigen
71 print(ergebnisse)
72
73 # Optional: Ergebnisse in eine neue Excel-Datei speichern

```



```

74 ergebnisse.to_excel('Prozentuale_Veraenderung_Fallzahlen.xlsx', index=False)
75 '''
76
77 ### Erklärung der Schritte:
78
79 1. **Einlesen aller Sheets:**
80 - Das Skript lädt die Excel-Datei 'Fallzahlen.xlsx' und liest alle Sheet-
    Namen.
81 - Es extrahiert die Jahreszahlen aus den Sheet-Namen, um sicherzustellen,
    dass die Daten chronologisch verarbeitet werden.
82 - Jeder Sheet wird in ein eigenes 'DataFrame' geladen und in einem
    Dictionary 'dataframes' gespeichert, wobei das Jahr als Schlüssel dient.
83
84 2. **Extraktion der 'Straftaten_insgesamt' für 'Berlin (PKS gesamt)':**
85 - Für jedes Jahr wird das entsprechende 'DataFrame' durchsucht, um die
    Zeile mit 'Bezirke' gleich 'Berlin (PKS gesamt)' zu finden.
86 - Der Wert aus der Spalte 'Straftaten_insgesamt' wird extrahiert und in
    dem Dictionary 'straftaten_gesamt' gespeichert.
87
88 3. **Berechnung der prozentualen Veränderung:**
89 - Die Jahreszahlen werden sortiert, um die chronologische Reihenfolge
    sicherzustellen.
90 - Für jedes Jahr ab dem zweiten wird die prozentuale Veränderung im
    Vergleich zum Vorjahr berechnet:
91     \[
92     \text{Veränderung} = \left( \frac{\text{Wert aktuell} - \text{Wert}}{\text{Wert vorher}} \right) \times 100
93     \]
94 - Die Ergebnisse werden im Dictionary 'prozentuale_veraenderung'
    gespeichert.
95
96 4. **Speichern der Ergebnisse:**
97 - Ein neues 'DataFrame' namens 'ergebnisse' wird erstellt, das die Jahre
    und die entsprechenden prozentualen Veränderungen enthält.
98 - Die Ergebnisse werden auf der Konsole angezeigt.
99 - Optional wird das Ergebnis auch in eine neue Excel-Datei '
    Prozentuale_Veraenderung_Fallzahlen.xlsx' gespeichert.
100
101 ### Hinweise:
102
103 - **Fehlerbehandlung:** Das Skript enthält grundlegende Fehlerbehandlungen, z
    .B. für Sheets, die nicht dem erwarteten Format entsprechen, oder wenn die
    Zeile 'Berlin (PKS gesamt)' nicht gefunden wird.
104
105 - **Anpassungen:** Falls die Excel-Datei weitere Sonderfälle enthält oder die
    Struktur variiert, müssen eventuell zusätzliche Anpassungen am Skript
    vorgenommen werden.
106
107 - **Sprachunterstützung:** Obwohl der Benutzer deutschsprachige Daten
    bereitstellt, werden die Spaltennamen im Skript entsprechend der
    bereitgestellten Daten verwendet. Stelle sicher, dass die Spaltennamen
    exakt übereinstimmen.

```

```

108
109 ### Beispielausgabe:
110
111 Angenommen, die Datei enthält Daten von 2014 bis 2020, könnte die Ausgabe wie
    folgt aussehen:
112
113 '''
114     Jahr   Prozentuale Veränderung (%)
115 0   2015                      5.23
116 1   2016                     -2.15
117 2   2017                      3.50
118 3   2018                      4.10
119 4   2019                      1.75
120 5   2020                     -0.85
121 '''
122
123 Diese Tabelle zeigt die prozentuale Veränderung der Straftaten insamt in
    Berlin im Vergleich zum Vorjahr für jedes Jahr.
124
125 Ich hoffe, dieses Skript hilft dir weiter! Falls du weitere Anpassungen benö
    tigtst oder Fragen hast, stehe ich gerne zur Verfügung.

```

Listing 121: AI Response Testfall 4 Prompt 3 Ausführung 1

```

1 Hier ist ein Python-Skript, das die von Ihnen beschriebenen Schritte ausführt
  . Stellen Sie sicher, dass Sie die Bibliothek 'pandas' installiert haben.
  Falls nicht, können Sie sie mit 'pip install pandas' installieren.
2
3 '''python
4 import pandas as pd
5
6 # Schritt 1: Alle Sheets der Excel-Datei einlesen
7 excel_datei = 'Fallzahlen.xlsx'
8 xls = pd.read_excel(excel_datei, sheet_name=None)
9
10 # Dictionary zum Speichern der 'Straftaten_insgesamt'-Werte pro Jahr
11 straftaten_dict = {}
12
13 # Schleife durch jedes Sheet
14 for sheet_name, df in xls.items():
15     # Extrahiere das Jahr aus dem Sheet-Namen, z.B. 'Fallzahlen_2014' -> 2014
16     try:
17         jahr = int(sheet_name.split('_')[-1])
18     except ValueError:
19         print(f"Sheet '{sheet_name}' entspricht nicht dem erwarteten Muster
    und wird übersprungen.")
20         continue
21
22     # Schritt 2: Wert der Spalte 'Straftaten_insgesamt' für 'Berlin (PKS
    gesamt)' extrahieren
23     try:
24         wert = df.loc[df['Bezirke'] == 'Berlin (PKS gesamt)', '
    Straftaten_insgesamt'].values[0]

```

```

25     straftaten_dict[jahr] = wert
26 except IndexError:
27     print(f"'Berlin (PKS gesamt)'" wurde in Sheet '{sheet_name}' nicht
gefunden.")
28     straftaten_dict[jahr] = None
29
30 # Umwandlung des Dictionaries in ein sortiertes DataFrame
31 straftaten_series = pd.Series(straftaten_dict).sort_index()
32 straftaten_df = straftaten_series.reset_index()
33 straftaten_df.columns = ['Jahr', 'Straftaten_insgesamt']
34
35 # Schritt 3: Prozentuale Veränderung zum Vorjahr berechnen
36 straftaten_df['Prozentuale_Veraenderung'] = straftaten_df['
Straftaten_insgesamt'].pct_change() * 100
37
38 # Optional: Prozentuale Veränderung auf zwei Dezimalstellen runden
39 straftaten_df['Prozentuale_Veraenderung'] = straftaten_df['
Prozentuale_Veraenderung'].round(2)
40
41 # Schritt 4: Ergebnisse in einem neuen DataFrame speichern
42 ergebnisse_df = straftaten_df[['Jahr', 'Prozentuale_Veraenderung']]
43
44 # Ausgabe der Ergebnisse
45 print(ergebnisse_df)
46
47 # Optional: Ergebnisse in eine neue Excel-Datei speichern
48 # ergebnisse_df.to_excel('Prozentuale_Veraenderung.xlsx', index=False)
49 '''
50
51 ### Erklärung der Schritte:
52
53 1. **Sheets einlesen:**
54 - Mit 'pd.read_excel' und 'sheet_name=None' werden alle Sheets der Excel-
Datei als ein Dictionary eingelesen, wobei die Sheet-Namen die Schlüssel
sind.
55
56 2. **Werte extrahieren:**
57 - Für jedes Sheet wird das Jahr aus dem Sheet-Namen extrahiert.
58 - Anschließend wird der Wert aus der Spalte ''Straftaten_insgesamt'' für
die Zeile ''Berlin (PKS gesamt)'' gesucht und im Dictionary '
straftaten_dict' gespeichert.
59
60 3. **Prozentuale Veränderung berechnen:**
61 - Das Dictionary wird in eine Pandas Series umgewandelt und nach Jahr
sortiert.
62 - Mit '.pct_change()' wird die prozentuale Veränderung zum Vorjahr
berechnet und mit 100 multipliziert, um den Wert in Prozent darzustellen.
63 - Die Ergebnisse werden auf zwei Dezimalstellen gerundet.
64
65 4. **Ergebnisse speichern:**
66 - Ein neues DataFrame 'ergebnisse_df' wird erstellt, das nur die Spalten '
'Jahr' und 'Prozentuale_Veraenderung' enthält.

```

```

67 - Die Ergebnisse werden anschließend ausgegeben. Optional können Sie die
    Ergebnisse auch in eine neue Excel-Datei speichern, indem Sie die
    auskommentierte Zeile am Ende des Skripts aktivieren.
68
69 ### Beispielausgabe:
70
71 '''
72     Jahr   Prozentuale_Veraenderung
73 0    2014                               NaN
74 1    2015                        5.23
75 2    2016                       -2.15
76 3    2017                        3.50
77 ...
78 '''
79
80 **Hinweis:** Der erste Eintrag für das Jahr 2014 hat keine prozentuale Verä
    nderung, da kein Vorjahr zum Vergleich vorhanden ist.
81
82 ---
83
84 Stellen Sie sicher, dass die Excel-Datei 'Fallzahlen.xlsx' im selben
    Verzeichnis wie das Skript liegt oder geben Sie den vollständigen Pfad zur
    Datei an. Wenn Sie die Ergebnisse in eine neue Excel-Datei speichern mö
    chten, entfernen Sie das Kommentarzeichen ('#') vor der letzten Zeile des
    Skripts.

```

Listing 122: AI Response Testfall 4 Prompt 3 Ausführung 2

```

1 Natürlich! Hier ist ein Python-Skript, das die beschriebenen Schritte durchfü
    hrt. Das Skript verwendet die 'pandas'-Bibliothek, um die Excel-Datei zu
    verarbeiten. Stelle sicher, dass du 'pandas' installiert hast ('pip
    install pandas openpyxl'), bevor du das Skript ausführst.
2
3 '''python
4 import pandas as pd
5
6 def main():
7     # Pfad zur Excel-Datei
8     excel_file = 'Fallzahlen.xlsx'
9
10    # Lade alle Sheet-Namen aus der Excel-Datei
11    xls = pd.ExcelFile(excel_file)
12    sheet_names = xls.sheet_names
13
14    # Dictionary zum Speichern der DataFrames
15    dataframes = {}
16
17    # Dictionary zum Speichern der 'Straftaten_insgesamt' Werte
18    strafaten_dict = {}
19
20    for sheet in sheet_names:
21        # Lese jedes Sheet in einen DataFrame
22        df = pd.read_excel(excel_file, sheet_name=sheet)

```

```

23     dataframes[sheet] = df
24
25     # Extrahiere das Jahr aus dem Sheet-Namen (angenommen, Format '
Fallzahlen_YYYY')
26     try:
27         year = int(sheet.split('_')[-1])
28     except ValueError:
29         print(f"Sheet-Name {sheet} entspricht nicht dem erwarteten Format
'Fallzahlen_YYYY'.")
30         continue
31
32     # Finde die Zeile für 'Berlin (PKS gesamt)'
33     berlin_row = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
34
35     if not berlin_row.empty:
36         # Extrahiere den Wert von 'Straftaten_insgesamt'
37         strafaten_value = berlin_row['Straftaten_insgesamt'].values[0]
38         strafaten_dict[year] = strafaten_value
39     else:
40         print(f"'Berlin (PKS gesamt)' nicht im Sheet {sheet} gefunden.")
41         strafaten_dict[year] = None
42
43     # Erstelle einen DataFrame aus den gesammelten Daten
44     strafaten_df = pd.DataFrame(list(strafaten_dict.items()), columns=['Jahr',
'Straftaten_insgesamt'])
45     strafaten_df.sort_values('Jahr', inplace=True)
46     strafaten_df.reset_index(drop=True, inplace=True)
47
48     # Berechne die prozentuale Veränderung zum Vorjahr
49     strafaten_df['Prozentuale_Veraenderung'] = strafaten_df['
Straftaten_insgesamt'].pct_change() * 100
50
51     # Optional: Runde die Veränderung auf 2 Dezimalstellen
52     strafaten_df['Prozentuale_Veraenderung'] = strafaten_df['
Prozentuale_Veraenderung'].round(2)
53
54     # Anzeige der Ergebnisse
55     print(strafaten_df)
56
57     # Optional: Speichern der Ergebnisse in eine neue Excel-Datei
58     output_file = 'Straftaten_Veraenderung.xlsx'
59     strafaten_df.to_excel(output_file, index=False)
60     print(f"Ergebnisse wurden in '{output_file}' gespeichert.")
61
62 if __name__ == "__main__":
63     main()
64
65
66 ### Erklärung der Schritte:
67
68 1. **Importieren der Bibliothek:**
69     '''python

```

```

70 import pandas as pd
71 '''
72
73 2. **Hauptfunktion definieren:**
74 - **Excel-Datei laden:**
75     '''python
76     excel_file = 'Fallzahlen.xlsx'
77     xls = pd.ExcelFile(excel_file)
78     sheet_names = xls.sheet_names
79     '''
80     Dies lädt alle Sheet-Namen aus der Excel-Datei.
81
82 - **Initialisieren von Dictionaries:**
83     '''python
84     dataframes = {}
85     strafaten_dict = {}
86     '''
87     'dataframes' speichert jeden DataFrame, 'strafaten_dict' speichert die
88     extrahierten Werte.
89
90 - **Durchlaufen der Sheets:**
91     '''python
92     for sheet in sheet_names:
93         df = pd.read_excel(excel_file, sheet_name=sheet)
94         dataframes[sheet] = df
95
96         try:
97             year = int(sheet.split('_')[-1])
98         except ValueError:
99             print(f"Sheet-Name {sheet} entspricht nicht dem erwarteten
100             Format 'Fallzahlen_YYYY'.")
101             continue
102
103         berlin_row = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
104
105         if not berlin_row.empty:
106             strafaten_value = berlin_row['Straftaten_insgesamt'].values[0]
107             strafaten_dict[year] = strafaten_value
108         else:
109             print(f"'Berlin (PKS gesamt)' nicht im Sheet {sheet} gefunden.")
110             strafaten_dict[year] = None
111     '''
112     - Jeder Sheet wird eingelesen und im 'dataframes' Dictionary gespeichert
113     .
114     - Das Jahr wird aus dem Sheet-Namen extrahiert.
115     - Die Zeile mit 'Bezirke' == 'Berlin (PKS gesamt)' wird gesucht und
116     der Wert von 'Straftaten_insgesamt' extrahiert.
117
118 - **Erstellen des Ergebnis-DatFrames:**
119     '''python
120     strafaten_df = pd.DataFrame(list(strafaten_dict.items()), columns=['Jahr
121     ', 'Straftaten_insgesamt'])

```

```

117     strafaten_df.sort_values('Jahr', inplace=True)
118     strafaten_df.reset_index(drop=True, inplace=True)
119     '''
120
121     - **Berechnen der prozentualen Veränderung:**
122     '''python
123     strafaten_df['Prozentuale_Veraenderung'] = strafaten_df['
Straftaten_insgesamt'].pct_change() * 100
124     strafaten_df['Prozentuale_Veraenderung'] = strafaten_df['
Prozentuale_Veraenderung'].round(2)
125     '''
126
127     - **Ergebnisse anzeigen und speichern:**
128     '''python
129     print(strafaten_df)
130     output_file = 'Straftaten_Veraenderung.xlsx'
131     strafaten_df.to_excel(output_file, index=False)
132     print(f"Ergebnisse wurden in '{output_file}' gespeichert.")
133     '''
134
135 3. **Skript ausführen:**
136     '''python
137     if __name__ == "__main__":
138         main()
139     '''
140
141 ### Zusätzliche Hinweise:
142
143 - **Fehlerbehandlung:**
144     - Das Skript gibt eine Warnung aus, wenn ein Sheet-Name nicht dem
    erwarteten Format entspricht oder wenn die spezifische Zeile nicht
    gefunden wird.
145
146 - **Speichern der Ergebnisse:**
147     - Die Ergebnisse werden nicht nur auf dem Bildschirm angezeigt, sondern
    auch in einer neuen Excel-Datei namens 'Straftaten_Veraenderung.xlsx'
    gespeichert.
148
149 - **Anpassungen:**
150     - Falls die Sheet-Namen ein anderes Muster haben, musst du die Methode zur
    Extraktion des Jahres ('year = int(sheet.split('_')[-1])') entsprechend
    anpassen.
151     - Stelle sicher, dass die Spaltennamen genau übereinstimmen (z.B. 'Bezirke
    ' und 'Straftaten_insgesamt').
152
153 ### Beispielausgabe:
154
155 Nach dem Ausführen des Skripts könnte die Ausgabe folgendermaßen aussehen:
156
157 '''
158     Jahr    Straftaten_insgesamt    Prozentuale_Veraenderung
159 0    2014                50000                NaN

```

```

160 1 2015          52000          4.00
161 2 2016          51000         -1.92
162 3 2017          53000          3.92
163 ...
164 '''
165
166 In diesem DataFrame repräsentiert jede Zeile ein Jahr und die entsprechende
    prozentuale Veränderung der Straftaten im Vergleich zum Vorjahr.
167
168 ---
169
170 Dieses Skript sollte dir dabei helfen, die gewünschten Analysen durchzuführen
    . Passe es nach Bedarf an deine spezifischen Anforderungen und
    Datenstrukturen an.

```

Listing 123: AI Response Testfall 4 Prompt 3 Ausführung 3

```

1 Hier ist ein Python-Skript, das die beschriebenen Schritte ausführt. Das
  Skript verwendet die 'pandas'-Bibliothek, um die Excel-Datei zu lesen und
  die gewünschten Berechnungen durchzuführen.
2
3 ### Voraussetzungen
4 Stellen Sie sicher, dass Sie die 'pandas'-Bibliothek installiert haben. Falls
  nicht, können Sie sie mit folgendem Befehl installieren:
5
6 ```bash
7 pip install pandas
8 ```
9
10 Zusätzlich benötigen Sie 'openpyxl' zum Lesen von Excel-Dateien im '.xlsx'-
  Format:
11
12 ```bash
13 pip install openpyxl
14 ```
15
16 ### Python-Skript
17
18 ```python
19 import pandas as pd
20
21 # 1. Lesen Sie alle Sheets der Excel-Datei ein und speichern Sie jedes Sheet
  in einem separaten DataFrame
22 excel_file = 'Fallzahlen.xlsx'
23 xls = pd.read_excel(excel_file, sheet_name=None) # Liest alle Sheets als ein
  Dictionary
24
25 # Dictionary zur Speicherung der 'Straftaten_insgesamt' Werte pro Jahr
26 straftaten_dict = {}
27
28 for sheet_name, df in xls.items():
29     # Extrahiere das Jahr aus dem Sheet-Namen, z.B. 'Fallzahlen_2014' -> 2014
30     try:

```



```

31     year = int(sheet_name.split('_')[-1])
32 except ValueError:
33     print(f"Sheet-Name '{sheet_name}' entspricht nicht dem erwarteten
34     Muster 'Fallzahlen_Jahr'. Übersprungen.")
35     continue
36
37 # 2. Extrahiere den Wert der Spalte 'Straftaten_insgesamt' für die Zeile
38 # 'Berlin (PKS gesamt)'
39 # Überprüfen Sie, ob die Spaltennamen korrekt sind. Möglicherweise müssen
40 # Sie 'Bezirke' anpassen.
41 # Hier wird angenommen, dass die Spalte mit Bezirken 'Bezirke' heißt
42 berlin_row = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
43
44 if not berlin_row.empty:
45     straf_value = berlin_row['Straftaten_insgesamt'].values[0]
46     straftaten_dict[year] = straf_value
47 else:
48     print(f"'Berlin (PKS gesamt)' nicht in Sheet '{sheet_name}' gefunden.
49     ")
50
51 # 3. Berechnen Sie die prozentuale Veränderung zum jeweiligen Vorjahr
52 # Sortieren der Jahre
53 sorted_years = sorted(straftaten_dict.keys())
54 percentage_changes = []
55
56 for i in range(1, len(sorted_years)):
57     current_year = sorted_years[i]
58     previous_year = sorted_years[i - 1]
59     current_value = straftaten_dict[current_year]
60     previous_value = straftaten_dict[previous_year]
61
62     # Prozentuale Veränderung berechnen
63     if previous_value != 0:
64         percent_change = ((current_value - previous_value) / previous_value)
65         * 100
66     else:
67         percent_change = None # Vermeiden von Division durch Null
68
69     percentage_changes.append({
70         'Jahr': current_year,
71         'Prozentuale Veränderung': percent_change
72     })
73
74 # 4. Speichern Sie die Ergebnisse in einem neuen DataFrame
75 result_df = pd.DataFrame(percentage_changes)
76
77 # Optional: Hinzufügen des ersten Jahres mit einer NaN für die Veränderung
78 result_df = pd.concat([
79     pd.DataFrame({'Jahr': [sorted_years[0]], 'Prozentuale Veränderung': [None]
80     }),
81     result_df
82 ], ignore_index=True)

```

```

77
78 # Anzeige des Ergebnisses
79 print(result_df)
80
81 # Optional: Speichern des Ergebnisses in eine neue Excel-Datei
82 result_df.to_excel('Prozentuale_Veraenderung_Straftaten.xlsx', index=False)
83 '''
84
85 ### Erklärung der Schritte
86
87 1. **Einlesen aller Sheets:**
88     - 'pd.read_excel' mit 'sheet_name=None' liest alle Sheets und speichert
      sie in einem Dictionary, wobei die Schlüssel die Sheet-Namen sind.
89
90 2. **Extraktion der relevanten Daten:**
91     - Für jedes Sheet wird das Jahr aus dem Namen extrahiert.
92     - Es wird die Zeile gesucht, in der die Spalte 'Bezirke' den Wert 'Berlin
      (PKS gesamt)' hat.
93     - Der Wert aus der Spalte 'Straftaten_insgesamt' wird extrahiert und im
      Dictionary 'straftaten_dict' gespeichert.
94
95 3. **Berechnung der prozentualen Veränderung:**
96     - Die Jahre werden sortiert, um sicherzustellen, dass die Berechnungen in
      chronologischer Reihenfolge erfolgen.
97     - Für jedes Jahr ab dem zweiten wird die prozentuale Veränderung im
      Vergleich zum Vorjahr berechnet.
98
99 4. **Speichern der Ergebnisse:**
100     - Die Ergebnisse werden in einem neuen DataFrame 'result_df' gespeichert,
      der die Jahre und die entsprechenden prozentualen Veränderungen enthält.
101     - Das Ergebnis wird angezeigt und optional in eine neue Excel-Datei '
      Prozentuale_Veraenderung_Straftaten.xlsx' gespeichert.
102
103 ### Beispielausgabe
104
105 Angenommen, Ihre Daten umfassen die Jahre 2014 bis 2016 mit folgenden
      fiktiven Werten:
106
107 | Jahr | Prozentuale Veränderung |
108 |-----|-----|
109 | 2014 | NaN |
110 | 2015 | 5.0 |
111 | 2016 | -3.8 |
112
113 Diese Tabelle zeigt, dass die Straftaten von 2014 auf 2015 um 5 % gestiegen
      und von 2015 auf 2016 um 3,8 % gesunken sind.
114
115 ### Hinweise
116
117 - Stellen Sie sicher, dass die Spaltennamen in Ihren Sheets genau mit den im
      Skript verwendeten Namen übereinstimmen ('Bezirke' und '
      Straftaten_insgesamt'). Andernfalls müssen Sie die Spaltennamen im Skript

```

```

    entsprechend anpassen.
118 - Falls Ihre Sheet-Namen ein anderes Muster haben oder zusätzliche
    Verarbeitung erfordern, passen Sie die Logik zur Extraktion des Jahres
    entsprechend an.
119 - Das Skript ignoriert Sheets, die nicht dem erwarteten Namensmuster
    entsprechen, und gibt eine Meldung aus.
120
121 Ich hoffe, dieses Skript erfüllt Ihre Anforderungen. Bei weiteren Fragen
    stehe ich gerne zur Verfügung!

```

Listing 124: AI Response Testfall 4 Prompt 3 Ausführung 4

```

1 Hier ist ein Python-Skript, das die von Ihnen beschriebenen Schritte ausführt
  . Das Skript verwendet die Bibliothek 'pandas', um die Excel-Datei zu
  verarbeiten und die erforderlichen Berechnungen durchzuführen.
2
3 Stellen Sie sicher, dass Sie 'pandas' installiert haben. Falls nicht, können
  Sie es mit 'pip' installieren:
4
5 ```bash
6 pip install pandas
7 ```
8
9 Hier ist das vollständige Python-Skript:
10
11 ```python
12 import pandas as pd
13
14 # 1. Lesen Sie alle Sheets der Excel-Datei ein
15 excel_file = 'Fallzahlen.xlsx'
16 all_sheets = pd.read_excel(excel_file, sheet_name=None)
17
18 # Liste zur Speicherung der Ergebnisse
19 daten_liste = []
20
21 # 2. Extrahieren Sie den Wert der Spalte 'Straftaten_insgesamt' für 'Berlin (
  PKS gesamt)'
22 for sheet_name, df in all_sheets.items():
23     # Extrahieren Sie das Jahr aus dem Sheet-Namen, z.B. 'Fallzahlen_2014' ->
      2014
24     try:
25         jahr = int(sheet_name.split('_')[-1])
26     except ValueError:
27         print(f"Das Sheet '{sheet_name}' hat keinen gültigen Jahresnamen und
        wird übersprungen.")
28         continue
29
30     # Suche nach der Zeile 'Berlin (PKS gesamt)'
31     berlin_data = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
32
33     if berlin_data.empty:
34         print(f"'Berlin (PKS gesamt)' wurde im Sheet '{sheet_name}' nicht
        gefunden.")

```

```

35         continue
36
37     # Extrahiere den Wert der Spalte 'Straftaten_insgesamt'
38     strftaten_insgesamt = berlin_data['Straftaten_insgesamt'].values[0]
39
40     daten_liste.append({'Jahr': jahr, 'Straftaten_insgesamt':
41         strftaten_insgesamt})
42
43 # 3. Erstellen Sie ein DataFrame aus den gesammelten Daten
44 daten_df = pd.DataFrame(daten_liste)
45
46 # Sortieren Sie das DataFrame nach Jahr aufsteigend
47 daten_df = daten_df.sort_values('Jahr').reset_index(drop=True)
48
49 # 4. Berechnen Sie die prozentuale Veränderung zum Vorjahr
50 daten_df['Prozentuale_Veraenderung'] = daten_df['Straftaten_insgesamt'].
51     pct_change() * 100
52
53 # Optional: Runden Sie die Prozentsätze auf zwei Dezimalstellen
54 daten_df['Prozentuale_Veraenderung'] = daten_df['Prozentuale_Veraenderung'].
55     round(2)
56
57 # 5. Erstellen Sie das Endergebnis DataFrame
58 ergebnis_df = daten_df[['Jahr', 'Prozentuale_Veraenderung']]
59
60 # Optional: Benennen Sie die Spalten auf Deutsch um
61 ergebnis_df.columns = ['Jahr', 'Prozentuale Veränderung (%)']
62
63 # 6. Anzeigen des Ergebnisses
64 print(ergebnis_df)
65
66 # 7. Speichern Sie das Ergebnis in einer neuen Excel-Datei (optional)
67 ergebnis_df.to_excel('Prozentuale_Veraenderung_Fallzahlen.xlsx', index=False)
68 '''
69
70 ### Erläuterung der Schritte:
71
72 1. **Einlesen aller Sheets:**
73     - Das Skript verwendet 'pd.read_excel' mit 'sheet_name=None', um alle
74     Sheets der Excel-Datei in ein Dictionary zu laden, wobei die Schlüssel die
75     Sheet-Namen und die Werte die jeweiligen DataFrames sind.
76
77 2. **Extrahieren der relevanten Daten:**
78     - Für jedes Sheet wird das Jahr aus dem Sheet-Namen extrahiert.
79     - Es wird nach der Zeile gesucht, in der die Spalte 'Bezirke' den Wert '
80     Berlin (PKS gesamt)' hat.
81     - Der Wert der Spalte 'Straftaten_insgesamt' wird extrahiert und in einer
82     Liste gespeichert.
83
84 3. **Erstellen und Sortieren des DataFrames:**
85     - Die gesammelten Daten werden in ein neues DataFrame 'daten_df'
86     umgewandelt.

```

```

79 - Das DataFrame wird nach dem Jahr aufsteigend sortiert, um
80 sicherzustellen, dass die prozentuale Veränderung korrekt berechnet wird.
81 4. Berechnung der prozentualen Veränderung:
82 - Die Funktion 'pct_change()' von pandas berechnet die prozentuale Verä
83 nderung zwischen den aufeinanderfolgenden Jahren.
84 - Das Ergebnis wird mit 100 multipliziert, um es in Prozent darzustellen.
85 - Optional werden die Prozentsätze auf zwei Dezimalstellen gerundet.
86 5. Erstellen des Endergebnis-DatFrames:
87 - Ein neues DataFrame 'ergebnis_df' wird erstellt, das nur die Spalten '
88 Jahr' und 'Prozentuale_Veraenderung' enthält.
89 - Die Spalten werden optional auf Deutsch umbenannt.
90 6. Anzeige und Speicherung des Ergebnisses:
91 - Das Endergebnis wird in der Konsole ausgegeben.
92 - Optional kann das Ergebnis auch in eine neue Excel-Datei gespeichert
93 werden.
94 ### Beispielausgabe:
95
96 Angenommen, die Daten enthalten die Jahre von 2014 bis 2016 mit
97 entsprechenden 'Straftaten_insgesamt' Werten. Die Ausgabe könnte wie folgt
98 aussehen:
99
100 '''
101   Jahr  Prozentuale Veränderung (%)
102 0  2014                      NaN
103 1  2015                      5.23
104 2  2016                     -3.45
105 '''
106 **Hinweis:**
107 - Der erste Eintrag hat keinen Vorjahr-Wert, daher ist die prozentuale Verä
108 nderung 'NaN'.
109 - Stellen Sie sicher, dass die Excel-Datei 'Fallzahlen.xlsx' im gleichen
110 Verzeichnis wie das Skript liegt oder geben Sie den vollständigen Pfad zur
111 Datei an.
112 - Passen Sie gegebenenfalls die Spaltennamen an, falls diese in der Excel-
113 Datei anders lauten.
114
115 Falls Sie weitere Anpassungen benötigen oder Fragen haben, lassen Sie es mich
116 wissen!

```

Listing 125: AI Response Testfall 4 Prompt 3 Ausführung 5

TODO: Abbildungsverzeichnis einfügen