



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

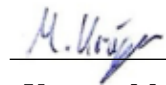
Erstgutachter: Prof. Dr.-Ing. Ingo Claßen

Zweitgutachter: Prof. Dr. Axel Hochstein

Eigenständigkeitserklärung

Ich versichere hiermit, dass ich die vorliegende Bachelorthesis selbstständig und ohne fremde Hilfe angefertigt und keine andere als die angegebene Literatur benutzt habe. Alle von anderen Autoren wörtlich übernommenen Stellen wie auch die sich an die Gedankengänge anderer Autoren eng anlehnenden Ausführungen meiner Arbeit sind besonders gekennzeichnet. Diese Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Berlin, den 15. 02. 2025

A handwritten signature in blue ink, appearing to read 'M. Krüger', is written above a horizontal line.

Krüger, Maurice

Abstract

Die rasante Entwicklung von Large Language Models (LLMs) hat neue Möglichkeiten für die automatisierte Code-Generierung eröffnet. Insbesondere für datenanalytische Aufgaben in Python könnten LLMs zukünftig manuelle Programmierung ersetzen oder ergänzen. Diese Arbeit untersucht die Eignung von LLMs zur Generierung von Python-Code für Datenanalyse-Aufgaben anhand einer empirischen Studie mit Kriminalitätsstatistiken der Stadt Berlin. Dazu wurden sechs spezifische Testfälle mit typischen datenanalytischen Aufgaben definiert. Die Code-Generierung erfolgte mithilfe von ChatGPT (GPTo1-mini) unter Anwendung von drei Prompting-Strategien: Zero-Shot, Instruction-based und Chain of Thought. Die generierten Skripte wurden hinsichtlich Korrektheit, Performanz, Wartbarkeit und Code-Qualität evaluiert. Die Ergebnisse zeigen, dass die Wahl der Prompting-Strategie entscheidend für die Erfolgsquote ist. Während Zero-Shot-Prompting nur eine 23%ige Erfolgsquote erreichte, erzielte Chain of Thought-Prompting mit 87% die besten Ergebnisse. Die generierten Skripte waren in vielen Fällen gut strukturiert, verständlich und performant, zeigten jedoch Schwächen bei komplexeren Aufgaben, insbesondere wenn unklare Spaltennamen oder unerwartete Datenstrukturen auftraten. Zusammenfassend verdeutlicht die Untersuchung, dass LLMs bereits leistungsfähige Werkzeuge für die Code-Generierung in der Datenanalyse sind, jedoch von präzisen Prompts und kontrollierten Evaluierungen profitieren.

1 Inhaltsverzeichnis

Inhaltsverzeichnis

1	Inhaltsverzeichnis	3
2	Abbildungsverzeichnis	6
3	Tabellenverzeichnis	7
4	Einleitung	8
4.1	Problemstellung und Forschungsfragen	8
4.2	Relevanz der Thematik	9
4.3	Zielsetzung	9
4.4	Aufbau der Arbeit	9
5	Grundlagen	9
5.1	Einführung Large Language Models	10
5.2	Einführung Python	11
5.2.1	Bedeutung und Bibliotheken	11
5.2.2	Typische Schritte einer Datenanalyse	11
5.3	Einführung automatisierte Code-Generierung	12
5.3.1	Funktionsweise und Vorteile	12
5.3.2	Herausforderungen und Grenzen	12
5.4	Prompting mit Sprachmodellen	12
6	LLMs in der Programmierung – aktueller Stand	13
6.1	Überblick und Vergleich von verschiedenen LLMs	13
6.2	Einsatzgebiete von LLMs in der Programmierung	15
6.3	Vergangene Studien und Arbeiten zur Code-Generierung	15
7	Ausgangsdaten und Testfallspezifikation	18
7.1	Überblick und Reduktion der Datengrundlage	18
7.2	Umwandlung in Pandas DataFrames	19
7.3	Testfälle und Vorgehen	19
7.3.1	Testfall 1: Sortierung und Ausgabe der Fallzahlen 2023	19
7.3.2	Testfall 2: Join aller Tabellen und „Bezirks-Topwert“	20
7.3.3	Testfall 3: Prozentuale Verteilung der Straftaten	20
7.3.4	Testfall 4: Zeitreihe über die Jahre 2014–2023	21
7.3.5	Testfall 5: Ermittlung des Unterbezirks mit den meisten Raubdelikten pro Oberbezirk	21
7.3.6	Testfall 6: Entwicklung der gefährlichsten Unterbezirke über die Jahre	22

8 Methodik	22
8.1 Vorgehensweise der Untersuchung	22
8.2 Testfälle der Datenanalyse	23
8.2.1 Testfall 1	23
8.2.2 Testfall 2	24
8.2.3 Testfall 3	24
8.2.4 Testfall 4	25
8.2.5 Testfall 5	26
8.2.6 Testfall 6	26
8.3 Auswertungskriterien	28
8.4 Verwendete Tools und Daten	28
9 Auswertung der Python-Code-Generierung zur Datenanalyse durch LLMs	29
9.1 Testfall 1: Sortierung und Ausgabe der Fallzahlen 2023	29
9.2 Testfall 2: Join aller Tabellen und Bezirks-Topwert	30
9.3 Testfall 3: Prozentuale Verteilung der Straftaten	32
9.4 Testfall 4: Zeitreihe über die Jahre 2014–2023	33
9.5 Testfall 5: Ermittlung des Unterbezirks mit den meisten Raubdelikten pro Oberbezirk	35
9.6 Testfall 6: Entwicklung der gefährlichsten Unterbezirke über die Jahre	36
9.7 Übersicht der Ergebnisse	38
9.8 Vergleich manuell erstellter Code	38
10 Limitationen, Fazit und Ausblick	39
10.1 Limitationen der Arbeit	39
10.2 Fazit	39
10.3 Ausblick	41
11 Anhang	42
11.1 Literaturverzeichnis	42
11.2 KI Verzeichnis	44
11.3 Quellcodeverzeichnis	45
11.3.1 Testfall 1	45
11.3.2 Testfall 2	61
11.3.3 Testfall 3	81
11.3.4 Testfall 4	100
11.3.5 Testfall 5	123
11.3.6 Testfall 6	146
11.3.7 Manuelle Skripte	169
11.3.8 Komplette Antworten der API Requests Testfall 1	173
11.3.9 Komplette Antworten der API Requests Testfall 2	217
11.3.10 Komplette Antworten der API Requests Testfall 3	258
11.3.11 Komplette Antworten der API Requests Testfall 4	301

11.3.12	Komplette Antworten der API Requests Testfall 5	350
11.3.13	Komplette Antworten der API Requests Testfall 6	397
11.3.14	Komplette Ausgaben der Skripte Testfall 1	446
11.3.15	Komplette Ausgaben der Skripte Testfall 2	467
11.3.16	Komplette Ausgaben der Skripte Testfall 3	485
11.3.17	Komplette Ausgaben der Skripte Testfall 4	499
11.3.18	Komplette Ausgaben der Skripte Testfall 5	511
11.3.19	Komplette Ausgaben der Skripte Testfall 6	526

2 Abbildungsverzeichnis

Abbildungsverzeichnis

1	Leistungsvergleich verschiedener Modellgrößen (nach Jiang et al. (2024), basierend auf 'A Survey on Large Language Models for Code Generation'[5]).	11
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'[5]).	14
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]).	16
4	Vergleich der Ergebnisse von <i>HumanEval</i> und <i>EvalPlus</i> (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]).	17

3 Tabellenverzeichnis

Tabellenverzeichnis

1	Ergebnisse von Testfall 1	29
2	Ergebnisse von Testfall 2	31
3	Ergebnisse von Testfall 3	32
4	Ergebnisse von Testfall 4	34
5	Ergebnisse von Testfall 5	35
6	Ergebnisse von Testfall 6	37
7	Übersicht der Pass@15-Ergebnisse pro Testfall und Prompting-Strategie . . .	38
8	KI-Verzeichnis	44

4 Einleitung

4.1 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, wie Liu et al. (2023)[1] und Chen et al. (2021)[2] zeigen.

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?

4.2 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.

4.3 Zielsetzung

Das Ziel dieser Arbeit ist es, herauszufinden, wie gut moderne LLMs 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, Performanz 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.

4.4 Aufbau der Arbeit

Nach dieser Einleitung (Kapitel 4) folgt in Kapitel 5 eine Darstellung der **Grundlagen**, dazu gehört eine Einführung in LLMs, die Programmiersprache Python, die automatisierte Code-Generierung und Prompting. Kapitel 6 gibt einen Überblick über den aktuellen Stand der Forschung, in dem verschiedene LLM-Modelle, Publikationen und Evaluationstechniken vorgestellt werden. Darauf aufbauend werden in Kapitel 7 die **Ausgangsdaten** und eine genaue **Testfallspezifikation** und in Kapitel 8 die **Methodik** der Arbeit erläutert. Kapitel 9 enthält dann die **Auswertung** der gewonnenen Daten sowie den Vergleich von durch ein LLM generierten und manuell geschriebenen Code. Kapitel 10.2 fasst die Ergebnisse zusammen, beantwortet die Forschungsfragen und gibt einen **Ausblick** auf weitere Entwicklungen. Schließlich enthält Kapitel 11 den **Anhang**, einschließlich Literaturverzeichnis und relevanter Dokumentationen.

5 Grundlagen

Im folgenden Kapitel werden die theoretischen und technischen Grundlagen vorgestellt, die für das Verständnis dieser Arbeit notwendig sind. Abschnitt 5.1 beschäftigt sich mit den LLMs, ihrer Funktionsweise und ihrer Bedeutung in der Code-Generierung. Danach wird in Abschnitt 5.2 das Potenzial der Programmiersprache Python für die Datenanalyse erläutert, bevor Abschnitt 5.3 das Konzept der automatisierten Code-Generierung behandelt und Abschnitt 5.4 eine kurze Einführung in Prompting gibt.

5.1 Einführung Large Language Models

Zhao et al.(2024)[4] erklären in ihrer Forschung, dass es sich bei LLMs um eine spezielle Klasse von vortrainierten Sprachmodellen (so genannten *Pre-trained Language Models* (PLMs)) mit einer besonders großen Anzahl an Parametern handelt – typischerweise im Bereich von mehreren zehn bis hunderten 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 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].

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.

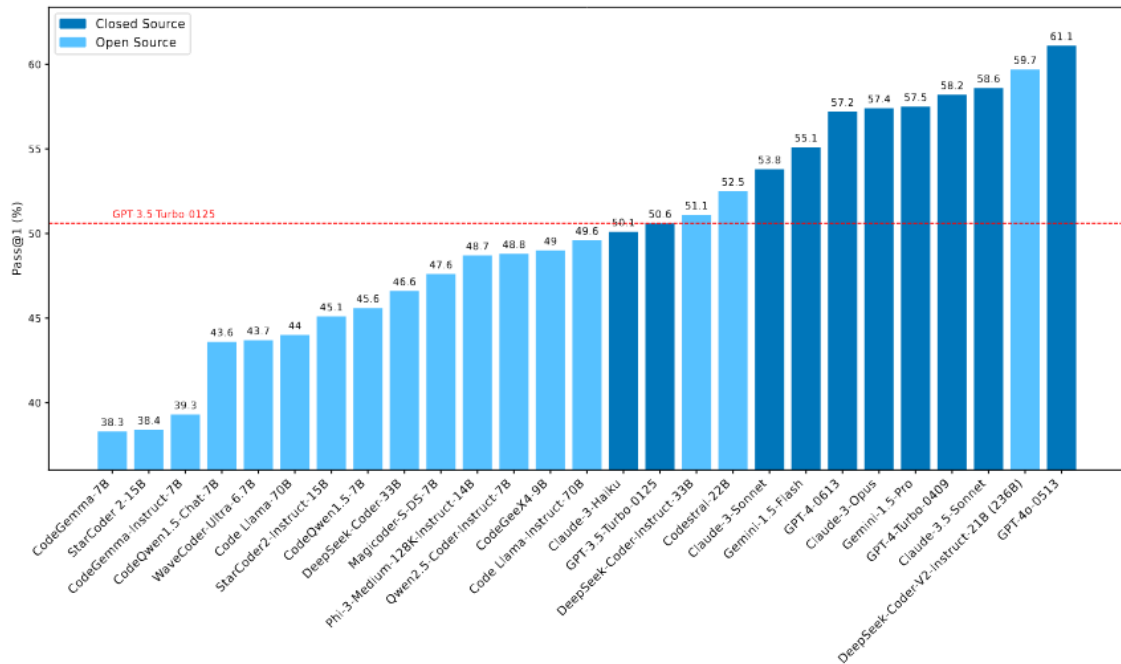


Abbildung 1: Leistungsvergleich verschiedener Modellgrößen (nach Jiang et al. (2024), basierend auf 'A Survey on Large Language Models for Code Generation'[5]).

5.2 Einführung Python

5.2.1 Bedeutung und Bibliotheken

Python ist dank seiner Syntax, aktiven Community und vielen hilfreichen Bibliotheken eine der am weitesten verbreiteten Sprachen für Datenanalyse[6]. 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].

5.2.2 Typische Schritte einer Datenanalyse

Die Grundschrte 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.

5.3 Einführung automatisierte Code-Generierung

5.3.1 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.

5.3.2 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[7].
- **Performanz:** Generierter Code ist nicht immer optimal hinsichtlich Laufzeit oder Speicherverbrauch[8].
- **Wartbarkeit:** Kommentare, klare Code-Struktur und Dokumentation fehlen häufig[7].
- **Fehleranfälligkeit:** Auch Code, der vorerst funktionsfähig erscheint, kann immer noch Bugs oder Sicherheitslücken enthalten[2, 7].

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.

5.4 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 mit LLMs direkt auf das bereits eingearbeitete Wissen des Modells zurückgegriffen und

steuert dadurch dessen Ausgabe durch passende Formulierungen[9]. 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)[10] 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.[10], 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[11].
- **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[12].

6 LLMs in der Programmierung – aktueller Stand

Die Entwicklung von 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 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].

6.1 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[13], StarCoder [14], CodeT5[15] oder CodeGen[16].

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[14] oder CodeGen[16] auch Open Source Alternativen.
- **Spezialisierung:** Einige Modelle sind speziell auf Code-Generierung abgestimmt (z.B. Code Llama, StarCoder[14]), wohingegen andere (z.B. ChatGPT) einen generellen Sprachkontext haben, um auch andere Fragen zu beantworten, der sich jedoch auch auf Code-Aufgaben anwenden lässt.

6.2 Einsatzgebiete von LLMs in der Programmierung

Die zunehmende Leistungsfähigkeit von 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, 15]. 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[15, 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 [17].

6.3 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.

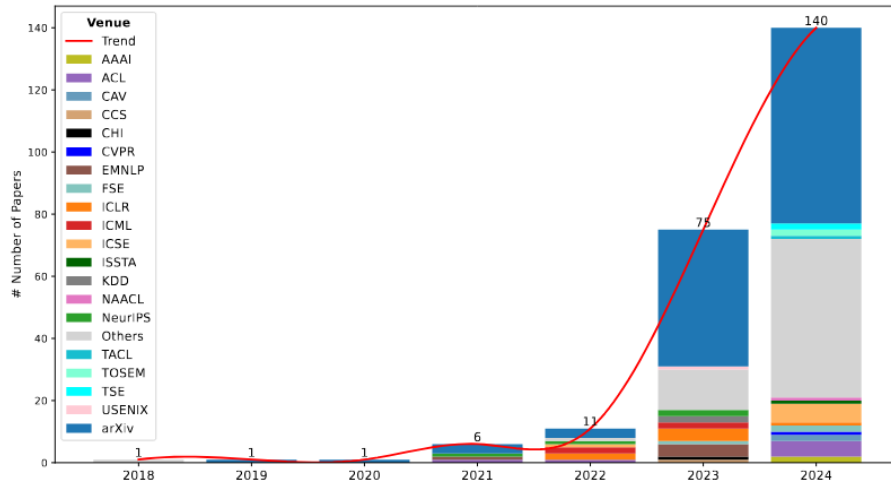


Abbildung 3: Übersicht der Verteilung von veröffentlichten Arbeiten zu LLMs und Software Engineering der letzten Jahren (nach Jiang et al. (2024), basierend auf 'A Survey on Large Language Models for Code Generation'[5]).

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. In Abbildung 4 sind die Ergebnisse von *HumanEval* und *EvalPlus* gegenübergestellt.
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 ₁ *	T ₁₀ *	T ₁₀₀ *
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 ⁴	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) [18]:

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.

7 Ausgangsdaten und Testfallspezifikation

7.1 Überblick und Reduktion der Datengrundlage

Die Datengrundlage für die empirische Untersuchung bilden Kriminalitätsstatistiken (sogenannte *Fallzahlen*) der Stadt Berlin[19]. 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.

7.2 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 (findet vor dem Einlesen mit Python statt).
2. **Bereinigung und Umbenennung:** Unnötige Spalten werden entfernt, Spaltennamen ggf. standardisiert (z. B. „Bezirk“, „Straftaten_insgesamt“)(findet vor dem Einlesen mit Python statt).
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.

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

7.3.1 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.
3. Sortierung nach `Straftaten_insgesamt` in absteigender oder aufsteigender Reihenfolge.
4. 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.

7.3.2 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:

1. Einlesen und Bereinigung der einzelnen `DataFrames` (2014–2023).
2. Zusammenführen nach dem `Bezirk`-Merkmal.
3. Aggregation der „`Straftaten_insgesamt`“ pro Jahr zu einer Gesamtzahl über alle Jahre.
4. 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)
...	...

7.3.3 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:

1. Einlesen des relevanten Sheets (z. B. `Fallzahlen_2023`).
2. 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_(%)
<i>Mitte</i>	<i>82 000</i>	<i>(z. B. 24,1%)</i>
<i>Neukölln</i>	<i>50 000</i>	<i>(z. B. 14,7%)</i>
...

7.3.4 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%
...	...

7.3.5 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

7.3.6 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

8 Methodik

8.1 Vorgehensweise der Untersuchung

In der Untersuchung soll geprüft werden, inwieweit LLMs in der Lage sind gängige Datenanalyse-Schritte auf Grundlage eines gegebenen Datensatzes durchzuführen. Hierbei wird ChatGPT

als aktueller Marktführer mit dem Sprachmodell GPTo1-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 4.1. Für die Vorgehensweise hierbei wird zuerst der verwendete Datensatz von Berlin Open Data, wie in Kapitel 7 beschrieben, heruntergebrochen und dann an das Modell übergeben und dazu, zu jeder verwendeten Prompting Strategie, eine Prompt verfasst. Diese Prompts können in Kapitel 8.2 eingesehen werden. Im Anschluss wird der Code ausgeführt, wobei seine Performanz mit Hilfe des Befehls `/usr/bin/time -v python` gemessen wird, und es werden manuelle Analysen durchgeführt um die Qualität und Effizienz des generierten Codes zu bewerten. Die genauen Auswertungskriterien sind in Kapitel 8.3 aufgeführt. Die Ergebnisse der Auswertung werden in Kapitel 9 detailliert dargestellt.

8.2 Testfälle der Datenanalyse

8.2.1 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.*

8.2.2 Testfall 2

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]*

8.2.3 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]*

8.2.4 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]`

8.2.5 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.*

8.2.6 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 zurueckgibt. 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:** *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.*

*Sortiere die Unterbezirke absteigend nach der aggregierten Anzahl an Straftaten.
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]

8.3 Auswertungskriterien

Üblicherweise werden Benchmarks, wie beispielsweise die zuvor erwähnten *HumanEval*[2] und *EvalPlus*[3], zur Bewertung von LLMs eingesetzt, um deren Leistungsfähigkeit im Bereich der Code Generierung zu vergleichen. Diese Benchmarks basieren jedoch oft auf Aufgaben, die einfache logische oder mathematische Schlussfolgerungen erfordern. In dieser Arbeit liegt der Fokus jedoch auf datenanalytischen Aufgaben, die eine komplexere Aufgabenstellung aus Datenverarbeitung und Analysen erfordern. Daher sind die bestehenden Benchmarks nicht direkt anwendbar, da sie keine realistischen Anforderungen an die datengetriebene Problemlösung stellen. Aufgrund der begrenzten Zeitvorgabe für diese Arbeit wurde auch davon abgesehen, selbst einen neuen Benchmark dafür einzuführen. Stattdessen wird die Leistung der Modelle anhand der in Kapitel 4.1 definierten Kriterien manuell vorgenommen.

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.

8.4 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[19].

9 Auswertung der Python-Code-Generierung zur Datenanalyse durch LLMs

9.1 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%

Performanz 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 150MB *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, Daten 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 auch gut erweiterbar, benötigten aber mehr Anpassungen zur Fehlerbehandlung.

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 Skripte zurück, die beinahe komplett 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 **Performanz** 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.

9.2 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 wie erwartet 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%)
Gesamt	5 / 15 (33% oder 100% je nach Bewertung)	10 / 15 (67%) oder 0%

Performanz 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 allen 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 **Performanz** gab es keine Probleme und alle Skripte waren effizient.

9.3 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.
- **Prompt 2 (Instruction Prompting):** Auch hier waren alle fünf Ausführungen erfolgreich. Die prozentualen Anteile wurden korrekt berechnet.
- **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 wurden.

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%)
Gesamt	14 / 15 (93%)	1 / 15 (7%)

Performanz 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, bei welchem an zwei Stellen die Python-Schlüsselwörter `if` und `not` ins Deutsche übersetzt wurden, in einer Ausführung scheiterte. 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 **Performanz** gab es keine Probleme, und alle Skripte waren effizient. Für Aufgaben dieses Umfangs lässt sich daher keine signifikante Verbesserung der Performanz durch die Wahl einer bestimmten Prompting-Strategie feststellen.

9.4 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).

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%)
Gesamt	11 / 15 (73%)	4 / 15 (27%)

Performanz 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.
- 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. Dies ist aber nicht auf die genutzte Prompting-Strategie zurückzuführen. Dennoch war statt **Chain of Thought Prompting Instruction Prompting** die am besten zu bewertende Prompting-Strategie.

In Bezug auf **Performanz** 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.

9.5 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:** Keine Ausführung war komplett erfolgreich. Hauptprobleme waren falsche Spaltennamen und fehlerhafte Gruppierung.
- **Instruction-based Prompting:** Keine Ausführung war erfolgreich, die Daten wurden zwar korrekt eingelesen und zugeordnet, jedoch wurden sie nicht als Dataframe ausgegeben.
- **Chain of Thought Prompting:** Alle fünf Ausführungen waren erfolgreich.

Tabelle 5: Ergebnisse von Testfall 5

Prompting-Strategie	Erfolgreiche Ausführungen	Fehlgeschlagene Ausführungen
Zero-Shot Prompting	0 / 5 (0%)	5 / 5 (100%)
Instruction-based Prompting	0 / 5 (0%)	5 / 5 (100%)
Chain of Thought Prompting	5 / 5 (100%)	0 / 5 (0%)
Gesamt	5 / 15 (33%)	10 / 15 (66%)

Performanz und Ressourcenverbrauch:

- Die Laufzeit betrug bei allen erfolgreichen Ausführungen zwischen **0,51 und 0,58 Sekunden**, was für die Berechnung der aggregierten Werte sehr effizient ist.
- Der Speicherverbrauch lag bei rund **150–155 MB Maximum Resident Set Size**, was für diesen Datensatz als angemessen betrachtet werden kann.
- Die CPU-Auslastung lag bei allen erfolgreichen Ausführungen zwischen **420 % und 460 %**, was auf eine effiziente Nutzung der verfügbaren Ressourcen hinweist.

Codequalität und Wartbarkeit:

- **Struktur:** Die Skripte von Prompt 2 und Prompt 3 waren klar strukturiert und enthielten die notwendigen Verarbeitungsschritte (*Daten einlesen, gruppieren, maximale Werte extrahieren, Daten speichern*). Die Skripte von Prompt 1 waren weniger strukturiert und enthielten teilweise ineffiziente Codeabschnitte.
- **Dokumentation:** Alle erfolgreichen Skripte waren gut kommentiert und erklärten die Berechnungen nachvollziehbar. Die besten Dokumentationen fanden sich in den Skripten von Prompt 3.
- **Erweiterbarkeit:** Die Skripte von Prompt 2 und 3 waren besonders gut erweiterbar, da sie eine klare Struktur aufwiesen und bereits eine robuste Fehlerbehandlung enthielten. Die Skripte von Prompt 1 waren ebenfalls erweiterbar, erforderten jedoch mehr Anpassungen, um fehlerhafte Gruppierungen zu korrigieren.

Fazit zu Testfall 5: Die Ergebnisse zeigen, dass die **Instruction Prompting**- und **Chain of Thought**-Strategien die zuverlässigsten Methoden zur Generierung von Python-Code für die Ermittlung des Unterbezirks mit den meisten Raubdelikten pro Oberbezirk sind. Beide Strategien erzielten eine Erfolgsquote von 100 % und lieferten gut strukturierte, robuste und gut dokumentierte Skripte.

Die **Zero-Shot Prompting**-Strategie hingegen war unzuverlässig, da sie in vier von fünf Fällen fehlerhafte Ergebnisse lieferte, insbesondere aufgrund fehlerhafter Spaltennamen und einer falschen Gruppierung der Daten.

In Bezug auf **Performanz** gab es keine signifikanten Unterschiede zwischen den Strategien, da alle erfolgreichen Skripte in unter einer Sekunde ausgeführt wurden und einen moderaten Speicherverbrauch aufwiesen. Für produktive Einsätze ist daher die Verwendung von **Instruction Prompting** oder **Chain of Thought Prompting** zu empfehlen, um sicherzustellen, dass der generierte Code zuverlässig, verständlich und erweiterbar ist.

9.6 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%)
Gesamt	3 / 15 (20%)	12 / 15 (80%)

Performanz 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. In den anderen Prompts war auch eine gute Dokumentation vorhanden, jedoch nicht so ausführlich.
- **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 **Performanz** 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.

9.7 Übersicht der Ergebnisse

Tabelle 7: Übersicht der Pass@15-Ergebnisse pro Testfall und Prompting-Strategie

Testfall	Zero-Shot	Instruction	Chain of Thought
1	0%	0%	100%
2	0%	0%	100%
3	100%	100%	80%
4	40%	100%	80%
5	0%	0%	100%
6	0%	0%	60%
Gesamt	23%	33%	87%

9.8 Vergleich manuell erstellter Code

In dieser Untersuchung werden die durch 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. 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.

Performanz 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 Performanz.

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.

10 Limitationen, Fazit und Ausblick

10.1 Limitationen der Arbeit

Die Untersuchung unterliegt einigen methodischen Einschränkungen, die bei der Interpretation der Ergebnisse berücksichtigt werden müssen:

- **Eingeschränkte Vergleichbarkeit mit Benchmarks:** Wie bereits im Kapitel Methodik (8) erwähnt, basieren übliche Benchmarks für LLMs vor allem auf logischen und mathematischen Aufgaben. Da datenanalytische Aufgaben komplexere Anforderungen haben, konnten die bestehenden Benchmarks für diese Forschung nicht genutzt werden und stattdessen basiert die Evaluierung auf einer manuellen Untersuchung anhand der in der Einleitung (4) gestellten Forschungsfragen.
- **Limitierte Interaktionsmöglichkeiten mit ChatGPT:** ChatGPT erlaubt keine direkten Dateianhänge, wie Excel-Dateien. Daher wurden die Prompts über die OpenAI API als Requests gesendet. Diese Requests sind jedoch mit Kosten verbunden und unterliegen einer Limitierung von Tokens. Aufgrund dessen wurde sich auf eine kleinere Anzahl an Testfällen beschränkt.
- **Zeitbeschränkungen:** Da die verfügbare Zeit für die Untersuchung begrenzt war, wurde sich auf eine kleinere Anzahl an Testfällen beschränkt. Zudem wurde ausschließlich ChatGPT mit GPTo1 untersucht, ohne weitere Sprachmodelle zu evaluieren.

10.2 Fazit

Die vorliegende Arbeit hat untersucht, wie gut sich moderne *Large Language Models* – exemplarisch repräsentiert durch ChatGPT mit dem Modell „GPTo1-mini“ – für die automatisierte Generierung von Python-Code im Kontext typischer Datenanalyseaufgaben eignen. Im Fokus standen vier typische Szenarien mit einem heruntergebrochenen Datensatz: das Sortieren und Filtern von Daten (Testfall1), das Zusammenführen und Aggregieren verschiedener

Tabellen (Testfall2), die Berechnung prozentualer Anteile (Testfall3), die Ermittlung von Zeitreihen mitsamt prozentualer Veränderung zum Vorjahr (Testfall4) und zwei Testfälle mit dem Ausgangsdatensatz: Zugehörigkeit untergeordneter Zeilen mit Sortierung (Testfall 5) und Zusammenführen verschiedener Tabellen, Sortieren und Identifizieren von unter- und übergeordneten Zeilen (Testfall 6). 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, mit einer insgesamt Erfolgsquote von 87%, kam es bei Zero-Shot-Ansätzen häufiger zu Syntaxfehlern, falschen Sheet-Namen oder fehlerhafter Ausfilterung bestimmter Zeilen, was zu einer Erfolgsquote von 23% führte. Insgesamt bestätigten sich die Erkenntnisse von Forschungen wie die von Wei et al. (2022)[12], dass *strukturierte Prompt-Anweisungen* (Instruction oder noch besser Chain of Thought) zu einer höheren Erfolgsquote führen als ein unpräziser Zero-Shot-Ansatz. Dies wurde besonders deutlich durch die Testfälle fünf und sechs, welche komplexere Anforderungen hatten. Jedoch muss hier auch festgehalten werden, dass viele Skripte, die als nicht erfolgreich gewertet wurden, nur kleine Differenzen zu den erwarteten Ergebnissen aufwiesen, wie zum Beispiel eine falsche Darstellung der Daten oder eine Zeile zu viel oder zu wenig, was jedoch das Ergebnis der anderen Daten nicht beeinflusste. Bei einer sanfteren Bewertung wären die Erfolgsquoten also höher.

- **Performanz:**

Bei allen Testfällen war die Ausführungszeit sehr kurz (teils nur wenige Zehntelsekunden bis wenige Sekunden), was für den überschaubaren Datensatz vollkommen ausreichend ist. Auch in den Testfällen fünf und sechs gab es hierbei keine signifikante Veränderung. Damit bestätigt sich, dass ein KI-basierter Code-Generator im Kontext kleiner bis mittlerer Datenanalysen in Sachen Laufzeit schon jetzt gute Ergebnisse liefern kann.

- **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. Dazu kommt, dass neben der Dokumentation im Code noch weitere Erklärungen zum Code als 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. Dies wird belegt durch die teils deutlich schlechteren Ergebnisse des *Zero-Shot*-Promptings.

- **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. Dies zeigte sich insbesondere in Testfall 6, wo Zero-Shot und Instruction Prompting sehr negativ abschnitten und selbst Chain of Thought Prompting eine nur 60%ige Erfolgsquote erreichte. Dies verdeutlicht, dass LLMs Schwierigkeiten mit komplexeren Datenstrukturen haben, insbesondere wenn sie sich über mehrere Tabellen erstrecken und es nicht normalisierte Spaltennamen gibt.

10.3 Ausblick

Während die Untersuchung gezeigt hat, dass LLMs großes Potenzial für die automatisierte Code-Generierung im Bereich der Datenanalyse haben, ergeben sich mehrere interessante Forschungsansätze, um die Leistungsfähigkeit dieser Modelle weiter zu verbessern und ihre praktischen Anwendungsmöglichkeiten zu erweitern.

Erweiterung bestehender Benchmarks für datenanalytische Aufgaben Die Evaluierung von LLMs für Code-Generierung basiert aktuell auf Benchmarks wie HumanEval oder EvalPlus, die einfache Programmieraufgaben bewerten. Diese Benchmarks erfassen jedoch nicht die spezifischen Herausforderungen datenanalytischer Aufgaben, die oft eine mehrstufige Datenverarbeitung erfordern. Zukünftige Forschungsarbeiten könnten bestehende Benchmarks erweitern oder neue Evaluationsmetriken für Datenanalyse-Aufgaben entwickeln, die unter anderem folgende Kriterien einbeziehen:

- Umgang mit komplexen Datenstrukturen und unvollständigen Datensätzen,
- Effizienz der generierten Skripte in Bezug auf Laufzeit und Speicherverbrauch,
- Code-Qualität, Wartbarkeit und Verständlichkeit in realistischen Szenarien.

Vergleich verschiedener LLMs für datenanalytische Code-Generierung Die Untersuchung beschränkte sich auf ChatGPT mit GPTo1, jedoch existieren zahlreiche weitere Modelle wie GPT-4 Turbo, Code Llama, StarCoder oder Claude, die ebenfalls für die Code-Generierung eingesetzt werden können. Ein Vergleich verschiedener Modelle hinsichtlich Genauigkeit, Korrektheit und Performanz wäre eine wertvolle Erweiterung zukünftiger Forschungen. Besonders spannend wäre die Frage, ob speziell für die Code-Generierung trainierte Modelle (z. B. Code Llama) besser für datenanalytische Aufgaben geeignet sind als generalisierte Modelle wie ChatGPT.

11 Anhang

11.1 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.
- [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=1qvx610Cu7>.
- [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] Sebastian Raschka, Joshua Patterson und Corey Nolet. *Machine Learning in Python: Main developments and technology trends in data science, machine learning, and artificial intelligence*. 2020. arXiv: 2002.04803 [cs.LG]. URL: <https://arxiv.org/abs/2002.04803>.
- [7] 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>.
- [8] 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>.
- [9] 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>.
- [10] Tom B. Brown u. a. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL]. URL: <https://arxiv.org/abs/2005.14165>.
- [11] 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>.
- [12] 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>.
- [13] Baptiste Rozière u. a. *Code Llama: Open Foundation Models for Code*. 2024. arXiv: 2308.12950 [cs.CL]. URL: <https://arxiv.org/abs/2308.12950>.

- [14] Raymond Li u. a. *StarCoder: may the source be with you!* 2023. arXiv: 2305.06161 [cs.CL]. URL: <https://arxiv.org/abs/2305.06161>.
- [15] 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>.
- [16] 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>.
- [17] 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>.
- [18] 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>.
- [19] Polizei Berlin. *Kriminalitätsatlas Berlin*. 2024. URL: <https://daten.berlin.de/datensaetze/kriminalitatsatlas-berlin>.

11.2 KI Verzeichnis

Tabelle 8: KI-Verzeichnis

KI-Tool	Teil der Arbeit (Seite)	Verwendungszweck
ChatGPT	Auswertung der Python-Code-Generierung zur Datenanalyse durch LLMs (S. 29-38)	Unterstützung der Ausformulierung der Auswertung der Testfälle und Auswertung des Ressourcenverbrauchs
ChatGPT	Abstract (S. 2)	Generierung des Abstracts (manuelle Anpassung des Abstracts danach)
ChatGPT	Einführung Large Language Models (S. 10)	Umformulierung mancher Aussagen
ChatGPT	Fazit und Ausblick (S. 38-41)	Hilfestellung bei der Formulierung mancher Abschnitte/Aussagen (Performanz, Grenzen und Vergleich verschiedener LLMs für datenanalytische Code-Generierung)
ChatGPT	LaTeX Code	Hilfe bei der Umwandlung von geschriebenem Text zu LaTeX Code; Hilfe bei einigen LaTeX Funktionen/-Fehlern

11.3 Quellcodeverzeichnis

11.3.1 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 ],)
29
30 response_file = path+"testcases/testcase1/prompt1/exec5/response.txt"
31 os.makedirs(os.path.dirname(response_file), exist_ok=True)
32
33 # Ausgabe des generierten Skripts
34 with open(response_file, "w") as file:
35     file.write(response.choices[0].message.content)
36
37 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
    sortierte_datei_pfad = 'Fallzahlen_sortiert.xlsx'
25     df_sortiert.to_excel(sortierte_datei_pfad, index=False)
26     print(f"\nDer sortierte DataFrame wurde in '{sortierte_datei_pfad}'
    gespeichert.")
27
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_schluesssel = [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_schluesssel)]
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_schluesel = [999900, 999999]
26 df_filtered = df[~df['LOR-Schlüssel'].isin(ausschluss_schluesel)]
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}'
12         gelesen.")
13         return df
14     except FileNotFoundError:
15         print(f"Fehler: Die Datei '{file_path}' wurde nicht gefunden.")
16         sys.exit(1)
17     except ValueError:
18         print(f"Fehler: Das Blatt '{sheet_name}' existiert nicht in '{
19         file_path}'")
20         sys.exit(1)
21     except Exception as e:
22         print(f"Ein unerwarteter Fehler ist aufgetreten: {e}")
23         sys.exit(1)
24
25 def validate_columns(df, required_columns):
26     """
27     Überprüft, ob die erforderlichen Spalten im DataFrame vorhanden sind.
28     """
29     missing_columns = [col for col in required_columns if col not in df.
30     columns]
31     if missing_columns:
32         print(f"Fehler: Die folgenden erforderlichen Spalten fehlen im
33         DataFrame: {missing_columns}")
34         sys.exit(1)
35     else:
36         print("Alle erforderlichen Spalten sind vorhanden.")
37
38 def sort_dataframe(df, sort_column, exclude_keys, key_column):
39     """
40     Sortiert den DataFrame nach 'sort_column' absteigend, wobei die Zeilen
41     mit 'exclude_keys'
42     am Ende verbleiben.
43     """
44     # Trennen der Zeilen, die ausgeschlossen werden sollen
45     df_excluded = df[df[key_column].isin(exclude_keys)]
46     df_to_sort = df[~df[key_column].isin(exclude_keys)]
47
48     # Sortieren der relevanten Zeilen
49     df_sorted = df_to_sort.sort_values(by=sort_column, ascending=False)
50
51     # Zusammenführen der sortierten und ausgeschlossenen Zeilen
52     df_final = pd.concat([df_sorted, df_excluded], ignore_index=True)

```

```

48     print("DataFrame erfolgreich sortiert.")
49     return df_final
50
51 def main():
52     # Parameter
53     file_path = 'Fallzahlen.xlsx'
54     sheet_name = 'Fallzahlen_2023'
55     sort_column = 'Straftaten_insgesamt'
56     key_column = 'LOR-Schlüssel'
57     exclude_keys = [999900, 999999]
58     output_file = 'Fallzahlen_sortiert.xlsx'
59
60     # Schritt 1: Daten einlesen
61     df = read_excel_file(file_path, sheet_name)
62
63     # Schritt 2: Validierung der erforderlichen Spalten
64     required_columns = [sort_column, key_column]
65     validate_columns(df, required_columns)
66
67     # Schritt 3: Sortierung durchführen
68     df_sorted = sort_dataframe(df, sort_column, exclude_keys, key_column)
69
70     # Optional: Weitere Überprüfung, ob alle Zeilen enthalten sind
71     if len(df_sorted) != len(df):
72         print("Warnung: Die Anzahl der Zeilen nach der Sortierung stimmt
73             nicht mit der ursprünglichen Anzahl überein.")
74
75     # Schritt 4: Ergebnis speichern
76     try:
77         df_sorted.to_excel(output_file, index=False)
78         print(f"Das sortierte DataFrame wurde erfolgreich in '{output_file}'
79             gespeichert.")
80     except Exception as e:
81         print(f"Fehler beim Speichern der Datei: {e}")
82         sys.exit(1)
83
84 if __name__ == "__main__":
85     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.

```

```

11
12 Returns:
13 - Sortierter Pandas DataFrame.
14 """
15 try:
16     # Schritt 1: Einlesen der Excel-Datei
17     df = pd.read_excel(file_path, sheet_name=sheet_name)
18     print(f"Excel-Datei '{file_path}' erfolgreich eingelesen.")
19
20     # Überprüfen, ob die notwendigen Spalten vorhanden sind
21     benötigte_spalten = ['LOR-Schlüssel', 'Straftaten_insgesamt']
22     fehlende_spalten = [spalte for spalte in benötigte_spalten if spalte
23 not in df.columns]
24     if fehlende_spalten:
25         raise ValueError(f"Fehlende Spalten in der Excel-Datei: {
26 fehlende_spalten}")
27
28     # Schritt 2: Trennen der Bezirke und Nicht-Bezirke
29     nicht_bezirke_keys = [999900, 999999]
30
31     # Sicherstellen, dass 'LOR-Schlüssel' als numerisch betrachtet wird
32     df['LOR-Schlüssel'] = pd.to_numeric(df['LOR-Schlüssel'], errors='
33 coerce')
34
35     bezirke_df = df[~df['LOR-Schlüssel'].isin(nicht_bezirke_keys)].copy()
36     nicht_bezirke_df = df[df['LOR-Schlüssel'].isin(nicht_bezirke_keys)].
37     copy()
38
39     # Schritt 3: Sortieren der Bezirke nach 'Straftaten_insgesamt'
40     absteigend
41     bezirke_df.sort_values(by='Straftaten_insgesamt', ascending=False,
42 inplace=True)
43
44     # Schritt 4: Zusammenfügen der sortierten Bezirke mit den Nicht-
45     Bezirken
46     sortiertes_df = pd.concat([bezirke_df, nicht_bezirke_df],
47 ignore_index=True)
48
49     print("Daten erfolgreich sortiert.")
50     return sortiertes_df
51
52 except FileNotFoundError:
53     print(f"Datei '{file_path}' wurde nicht gefunden.")
54 except ValueError as ve:
55     print(f"Wertfehler: {ve}")
56 except Exception as e:
57     print(f"Ein unerwarteter Fehler ist aufgetreten: {e}")
58
59 # Anwendung des Skripts
60 if __name__ == "__main__":
61     datei_pfad = 'Fallzahlen.xlsx'
62     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.
        columns]
26     if missing_columns:
27         print(f"Fehler: Fehlende Spalten in den Daten: {missing_columns}")
28         sys.exit(1)
29
30     # Sicherstellen, dass 'LOR-Schlüssel' numerisch ist
31     try:
32         df['LOR-Schlüssel'] = pd.to_numeric(df['LOR-Schlüssel'])
33     except ValueError:
34         print("Fehler: Die Spalte 'LOR-Schlüssel' enthält nicht-numerische
        Werte.")
35         sys.exit(1)

```

```

36
37 # Schritt 2: Trennen der Bezirke und der auszuschließenden Schlüssel
38 df_bezirke = df[~df['LOR-Schlüssel'].isin(lor_keys_to_exclude)]
39 df_excluded = df[df['LOR-Schlüssel'].isin(lor_keys_to_exclude)]
40
41 # Schritt 3: Sortieren der Bezirke nach 'Straftaten_insgesamt' absteigend
42 if 'Straftaten_insgesamt' not in df_bezirke.columns:
43     print("Fehler: Die Spalte 'Straftaten_insgesamt' ist nicht in den
44     Daten vorhanden.")
45     sys.exit(1)
46
47 try:
48     df_bezirke_sorted = df_bezirke.sort_values(by='Straftaten_insgesamt',
49     ascending=False)
50 except Exception as e:
51     print(f"Fehler beim Sortieren der Daten: {e}")
52     sys.exit(1)
53
54 # Schritt 4: Zusammenführen der sortierten Bezirke mit den
55 # ausgeschlossenen Zeilen
56 df_final = pd.concat([df_bezirke_sorted, df_excluded], ignore_index=True)
57
58 # Optional: Ausgabe oder Speicherung des finalen DataFrames
59 # Beispiel: Anzeigen der ersten 5 Zeilen
60 print("Sortiertes DataFrame:")
61 print(df_final.head())
62
63 # Beispiel: Speichern in eine neue Excel-Datei
64 output_file = 'Fallzahlen_sortiert.xlsx'
65 try:
66     df_final.to_excel(output_file, index=False)
67     print(f"Das sortierte DataFrame wurde erfolgreich in '{output_file}'
68     gespeichert.")
69 except Exception as e:
70     print(f"Fehler beim Speichern der Datei '{output_file}': {e}")
71     sys.exit(1)
72
73 if __name__ == "__main__":
74     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:

```

```

10     - file_path: Path to the Excel file.
11     - sheet_name: Name of the sheet to read.
12
13     Returns:
14     - A sorted pandas DataFrame.
15     """
16     try:
17         # Read the Excel sheet
18         df = pd.read_excel(file_path, sheet_name=sheet_name)
19         print(f"Successfully read the sheet '{sheet_name}' from '{file_path}'")
20     except FileNotFoundError:
21         print(f"Error: The file '{file_path}' was not found.")
22         return pd.DataFrame()
23     except ValueError:
24         print(f"Error: The sheet '{sheet_name}' does not exist in '{file_path}'")
25         return pd.DataFrame()
26     except Exception as e:
27         print(f"An unexpected error occurred while reading the Excel file: {e}")
28         return pd.DataFrame()
29
30     # Check if necessary columns exist
31     required_columns = ['LOR-Schlüssel', 'Straftaten_insgesamt']
32     missing_columns = [col for col in required_columns if col not in df.columns]
33     if missing_columns:
34         print(f"Error: The following required columns are missing from the data: {missing_columns}")
35         return pd.DataFrame()
36
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 order
46     try:
47         included_sorted_df = included_df.sort_values(by='Straftaten_insgesamt', ascending=False)
48         print("Successfully sorted the included data by 'Straftaten_insgesamt' in descending order.")
49     except KeyError:
50         print("Error: The column 'Straftaten_insgesamt' does not exist in the data.")
51         return pd.DataFrame()
52     except Exception as e:

```

```

53     print(f"An unexpected error occurred during sorting: {e}")
54     return pd.DataFrame()
55
56     # Concatenate the sorted included data with the excluded data
57     sorted_df = pd.concat([included_sorted_df, excluded_df], ignore_index=
True)
58     print("Successfully concatenated the excluded rows at the end of the
sorted data.")
59
60     return sorted_df
61
62 if __name__ == "__main__":
63     # Define the Excel file path and sheet name
64     excel_file = 'Fallzahlen.xlsx'
65     sheet = 'Fallzahlen_2023'
66
67     # Get the sorted DataFrame
68     sorted_fallzahlen_df = sort_fallzahlen_excel(excel_file, sheet)
69
70     # Check if the DataFrame is not empty before proceeding
71     if not sorted_fallzahlen_df.empty:
72         print("Sorted DataFrame:")
73         print(sorted_fallzahlen_df)
74         # Optionally, you can save the sorted DataFrame to a new Excel file
or another format
75         # sorted_fallzahlen_df.to_excel('Sorted_Fallzahlen.xlsx', index=False
)
76     else:
77         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 '
Straftaten_insgesamt' absteigend,
6     wobei die Zeilen mit den LOR-Schlüsseln 999900 und 999999 am Ende des
DataFrames stehen.
7
8     Parameters:
9     - file_path: Pfad zur Excel-Datei.
10    - sheet_name: Name des Sheets, das eingelesen werden soll.
11
12    Returns:
13    - Ein sortierter Pandas DataFrame.
14    """
15    try:
16        # Schritt 1: Einlesen der Excel-Datei
17        df = pd.read_excel(file_path, sheet_name=sheet_name)
18        print(f"Erfolgreich das Sheet '{sheet_name}' aus '{file_path}'
eingelesen.")

```

```

19     except FileNotFoundError:
20         raise FileNotFoundError(f"Die Datei '{file_path}' wurde nicht
gefunden.")
21     except ValueError:
22         raise ValueError(f"Das Sheet '{sheet_name}' existiert nicht in '{
file_path}'".)
23     except Exception as e:
24         raise Exception(f"Ein unerwarteter Fehler ist beim Einlesen der Excel
-Datei aufgetreten: {e}")
25
26     # Überprüfen, ob die notwendigen Spalten vorhanden sind
27     required_columns = ['LOR-Schlüssel', 'Straftaten_insgesamt']
28     missing_columns = [col for col in required_columns if col not in df.
columns]
29     if missing_columns:
30         raise KeyError(f"Die folgenden erforderlichen Spalten fehlen im
DataFrame: {missing_columns}")
31     else:
32         print("Alle erforderlichen Spalten sind vorhanden.")
33
34     # Schritt 2: Separiere die Bezirke und die nicht zuzuordnenden Einträge
35     # Definiere die LOR-Schlüssel, die nicht sortiert werden sollen
36     exclude_lor_keys = [999900, 999999]
37
38     # Konvertiere 'LOR-Schlüssel' zu integer, falls nötig
39     if df['LOR-Schlüssel'].dtype != 'int64' and df['LOR-Schlüssel'].dtype !=
'int32':
40         try:
41             df['LOR-Schlüssel'] = df['LOR-Schlüssel'].astype(int)
42             print("'LOR-Schlüssel' erfolgreich in Integer konvertiert.")
43         except ValueError:
44             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: Konkateniere 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}")

```

Listing 16: Skript Testfall 1 Prompt 3 Ausführung 5

11.3.2 Testfall 2

```

1 from openai import OpenAI
2 import os
3 import pandas as pd
4
5 path=os.getenv("THEISIS_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 df = pd.read_excel(file, sheet_name='Fallzahlen_2023')
14
15 # Prompt erstellen: CSV-Daten einbetten + Anweisung

```

```

16 prompt = f"""
17 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: {
    df}
18 """
19
20 # Anfrage an die OpenAI Chat-API
21 response = client.chat.completions.create(model="o1-mini",
22 messages=[
23 {"role": "user", "content": prompt}
24 ],)
25
26 response_file = path+"testcases/testcase2/prompt1/exec5/response.txt"
27 os.makedirs(os.path.dirname(response_file), exist_ok=True)
28 # Ausgabe des generierten Skripts
29 with open(response_file, "w") as file:
30     file.write(response.choices[0].message.content)
31
32 print(f"Response wurde in {response_file} gespeichert.")

```

Listing 17: OpenAI API Request Skript

```

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)

```

```

28     # Prüfen, ob die Spalte 'Straftaten_insgesamt' existiert
29     if 'Straftaten_insgesamt' not in kombiniertes_df.columns:
30         raise ValueError("Die Spalte 'Straftaten_insgesamt' wurde in den
Daten nicht gefunden.")
31
32     # Sortieren nach 'Straftaten_insgesamt' in absteigender Reihenfolge
33     sortiertes_df = kombiniertes_df.sort_values(by='Straftaten_insgesamt',
, ascending=False).reset_index(drop=True)
34
35     return sortiertes_df
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}")
41
42 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 18: 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
30 # Kombinieren aller relevanten Daten
31 if daten_liste:
32     alle_daten = pd.concat(daten_liste)
33 else:
34     print("Keine relevanten Daten gefunden.")
35     exit(1)
36
37 # Aggregation der Straftaten pro Bezirk
38 aggregierte_daten = alle_daten.groupby('Bezirke', as_index=False)['
39     Straftaten_insgesamt'].sum()
40
41 # Sortieren der Bezirke nach der Anzahl der Straftaten (absteigend)
42 aggregierte_daten_sortiert = aggregierte_daten.sort_values(by='
43     Straftaten_insgesamt', ascending=False).reset_index(drop=True)
44
45 # Optional: Anzeige des resultierenden DataFrames
46 print(aggregierte_daten_sortiert)
47
48 # Optional: Speichern des DataFrames in eine neue Excel-Datei
49 # aggregierte_daten_sortiert.to_excel('Aggregierte_Fallzahlen.xlsx', index=
50     False)

```

Listing 19: 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 # kombiniertes_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 20: 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 21: 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 22: 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.")

```

```

36
37     # Sortieren Sie den DataFrame nach 'Straftaten_insgesamt' in absteigender
    Reihenfolge
38     sorted_df = filtered_df.sort_values(by='Straftaten_insgesamt', ascending=
False).reset_index(drop=True)
39     print("DataFrame wurde nach 'Straftaten_insgesamt' sortiert.")
40
41     return sorted_df
42
43 if __name__ == "__main__":
44     # Pfad zur Excel-Datei
45     excel_pfad = 'Fallzahlen.xlsx'
46
47     # Rufen Sie die Funktion auf und speichern Sie das Ergebnis in 'final_df'
48     final_df = zusammenfuegen_und_sortieren(excel_pfad)
49
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('Zusammengefuegte_Fallzahlen.xlsx', index=False)

```

Listing 23: 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
    zurück.
7
8     Parameters:
9         file_path (str): Der Pfad zur Excel-Datei.
10
11     Returns:
12         pd.DataFrame: Der verarbeitete DataFrame.
13     """
14     try:
15         # Lade alle Sheets in ein Dictionary von DataFrames
16         all_sheets = pd.read_excel(file_path, sheet_name=None, dtype={'LOR-
    Schlüssel': str})
17
18         # Liste zum Speichern der DataFrames
19         df_list = []
20
21         for sheet_name, df in all_sheets.items():
22             print(f"Lade Sheet: {sheet_name} mit {len(df)} Zeilen.")
23             df_list.append(df)
24
25         # Kombiniere alle DataFrames in einen einzigen DataFrame
26         combined_df = pd.concat(df_list, ignore_index=True)
27         print(f"Gesamtanzahl der Zeilen nach dem Zusammenführen: {len(

```

```

combined_df})")
28
29     # Entferne Zeilen mit LOR-Schlüssel 999900 und 999999
30     filtered_df = combined_df[~combined_df['LOR-Schlüssel'].isin([999900,
31     999999])]
32     print(f"Anzahl der Zeilen nach dem Filtern: {len(filtered_df)}")
33
34     # Sortiere nach 'Straftaten_insgesamt' absteigend
35     sorted_df = filtered_df.sort_values(by='Straftaten_insgesamt',
36     ascending=False)
37     print("Daten wurden nach 'Straftaten_insgesamt' sortiert.")
38
39     # Optional: Setze den Index zurück
40     sorted_df.reset_index(drop=True, inplace=True)
41
42     return sorted_df
43
44 except FileNotFoundError:
45     print(f"Die Datei {file_path} wurde nicht gefunden.")
46 except Exception as e:
47     print(f"Ein Fehler ist aufgetreten: {e}")
48
49 if __name__ == "__main__":
50     # Pfad zur Excel-Datei
51     excel_file = 'Fallzahlen.xlsx'
52
53     # Verarbeite die Excel-Datei
54     final_df = load_and_process_excel(excel_file)
55
56     if final_df is not None:
57         # Zeige die ersten paar Zeilen des finalen DataFrames
58         print("\nErgebnis:")
59         print(final_df.head())
60
61         # Optional: Speichere den DataFrame in eine neue Excel- oder CSV-
62         Datei
63         # final_df.to_excel('Zusammengefuegte_Fallzahlen.xlsx', index=False)
64         # final_df.to_csv('Zusammengefuegte_Fallzahlen.csv', index=False)

```

Listing 24: 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
19             fehlen.")
20         continue
21
22     # Ausschließen der unerwünschten LOR-Schlüssel
23     df_filtered = df[~df['LOR-Schlüssel'].isin(exclude_lor_keys)].copy()
24
25     # Optional: Hinzufügen einer Spalte zur Kennzeichnung des Sheets
26     df_filtered['Sheet'] = sheet_name
27
28     filtered_dfs.append(df_filtered)
29
30 # Überprüfen, ob es DataFrames zum Zusammenführen gibt
31 if not filtered_dfs:
32     raise ValueError("Keine gültigen Daten zum Zusammenführen gefunden.")
33
34 # Zusammenführen aller DataFrames auf Basis von 'LOR-Schlüssel' und 'Bezirke'
35 # Falls es gemeinsame Spalten außer den Schlüsseln gibt, werden sie mit
36 # Suffixen versehen
37 from functools import reduce
38
39 # Füge die DataFrames schrittweise zusammen
40 merged_df = reduce(lambda left, right: pd.merge(left, right, on=['LOR-Schlüssel', 'Bezirke'], how='outer', suffixes=('', '_dup')), filtered_dfs)
41
42 # Optional: Entfernen von doppelten Spalten, die durch die Zusammenführung
43 # entstehen könnten
44 merged_df = merged_df.loc[:, ~merged_df.columns.duplicated()]
45
46 # Sortieren nach 'Straftaten_insgesamt' absteigend
47 if 'Straftaten_insgesamt' in merged_df.columns:
48     sorted_df = merged_df.sort_values(by='Straftaten_insgesamt', ascending=False)
49 else:
50     print("'Straftaten_insgesamt' Spalte nicht gefunden. Sortierung übersprungen.")
51     sorted_df = merged_df
52
53 # Reset des Indexes
54 sorted_df.reset_index(drop=True, inplace=True)
55
56 # Ergebnis anzeigen
57 print(sorted_df)
58
59 # Optional: Speichern des sortierten DataFrames in eine neue Excel-Datei
60 sorted_df.to_excel('Zusammengefuegte_Fallzahlen.xlsx', index=False)

```

Listing 25: 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-
6     Schlüssel,
7     fasst die Daten der Bezirke zusammen, sortiert sie nach
8     Straftaten_insgesamt
9     und gibt das resultierende DataFrame zurück.
10
11     Parameters:
12     - excel_file (str): Pfad zur Excel-Datei (z.B. 'Fallzahlen.xlsx')
13
14     Returns:
15     - pd.DataFrame: Gefiltertes und sortiertes DataFrame
16     """
17
18     # Definiere die unerwünschten LOR-Schlüssel
19     exclude_lor_keys = [999900, 999999]
20
21     try:
22         # Lade alle Sheets in ein Dictionary von DataFrames
23         sheets_dict = pd.read_excel(excel_file, sheet_name=None)
24         print(f"Es wurden {len(sheets_dict)} Sheets gefunden und geladen.")
25     except FileNotFoundError:
26         print(f"Die Datei '{excel_file}' wurde nicht gefunden.")
27         return None
28     except Exception as e:
29         print(f"Ein Fehler ist beim Lesen der Excel-Datei aufgetreten: {e}")
30         return None
31
32     # Liste zur Speicherung gefilterter DataFrames
33     filtered_dfs = []
34
35     # Iteriere über jedes Sheet und filtere die unerwünschten Zeilen
36     for sheet_name, df in sheets_dict.items():
37         print(f"Verarbeite Sheet: {sheet_name}")
38
39         # Überprüfe, ob die erforderlichen Spalten vorhanden sind
40         required_columns = ['LOR-Schlüssel', 'Bezirke', 'Straftaten_insgesamt']
41
42         if not all(col in df.columns for col in required_columns):
43             print(f"Sheet '{sheet_name}' fehlt eine oder mehrere
44             erforderliche Spalten. Überspringe dieses Sheet.")
45             continue
46
47         # Filtere die unerwünschten LOR-Schlüssel
48         df_filtered = df[~df['LOR-Schlüssel'].isin(exclude_lor_keys)].copy()
49         print(f"Anzahl der Bezirke nach Filtern in Sheet '{sheet_name}': {len
```



```

(df_filtered)}")
46
47     # Optional: Falls du weitere Daten aggregieren möchtest, kannst du
hier weitere Schritte hinzufügen
48
49     filtered_dfs.append(df_filtered)
50
51     if not filtered_dfs:
52         print("Keine gültigen Daten zum Zusammenfügen gefunden.")
53         return None
54
55     # Füge alle gefilterten DataFrames zusammen
56     combined_df = pd.concat(filtered_dfs, ignore_index=True)
57     print(f"Gesamtanzahl der Zeilen nach dem Zusammenfügen: {len(combined_df)}")
58
59     # Gruppieren die Daten nach 'LOR-Schlüssel' und 'Bezirke' und summiere '
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 26: 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
6   Sheets,
7   entfernt bestimmte LOR-Schlüssel, fasst die Daten zusammen, sortiert sie
8   nach
9   'Straftaten_insgesamt' und gibt den finalen DataFrame zurück.
10
11   :param file_path: Pfad zur Excel-Datei
12   :return: Gefilterter und sortierter Pandas DataFrame
13   """
14   # Lese die Excel-Datei
15   try:
16       xls = pd.ExcelFile(file_path)
17   except FileNotFoundError:
18       print(f"Die Datei {file_path} wurde nicht gefunden.")
19       return None
20   except Exception as e:
21       print(f"Ein Fehler ist aufgetreten: {e}")
22       return None
23
24   # Liste zur Speicherung der einzelnen DataFrames
25   df_list = []
26
27   # Iteriere über alle Sheets
28   for sheet_name in xls.sheet_names:
29       try:
30           df = pd.read_excel(xls, sheet_name=sheet_name)
31
32           # Überprüfe, ob die notwendigen Spalten vorhanden sind
33           required_columns = ['LOR-Schlüssel', 'Bezirke', '
Straftaten_insgesamt']
34           if not all(col in df.columns for col in required_columns):
35               print(f"Ein oder mehrere erforderliche Spalten fehlen im
Sheet '{sheet_name}'.")
36               continue
37
38           # Entferne die Zeilen mit unerwünschten LOR-Schlüsseln
39           df_filtered = df[~df['LOR-Schlüssel'].isin([999900, 999999])]
40
41           # Wähle nur die benötigten Spalten
42           df_selected = df_filtered[required_columns]
43
44           # Füge eine neue Spalte für das Sheet hinzu (optional, falls benö
tigt)
45           df_selected['Sheet'] = sheet_name
46
47           # Füge den DataFrame der Liste hinzu
48           df_list.append(df_selected)
49
50       except Exception as e:
51           print(f"Fehler beim Verarbeiten des Sheets '{sheet_name}': {e}")
52           continue

```

```

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     # Gruppieren 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 27: 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
22                 Spalten.")
23             continue
24
25         # Setzen von 'LOR-Schlüssel' und 'Bezirke' als Index
26         df = df.set_index(['LOR-Schlüssel', 'Bezirke'])
27         df_liste.append(df)
28
29     if not df_liste:
30         print("Keine gültigen Sheets zum Verarbeiten gefunden.")
31         return
32
33     # Alle DataFrames zusammenfügen und numerische Spalten akkumulieren
34     zusammengefuegt_df = pd.concat(df_liste)
35     # Gruppieren nach den Index-Spalten und summieren der numerischen Werte
36     akkumuliert_df = zusammengefuegt_df.groupby(level=['LOR-Schlüssel', '
37         Bezirke']).sum().reset_index()
38
39     # 3. DataFrame sortieren nach 'Straftaten_insgesamt', spezielle Zeilen am
40     Ende platzieren
41     # Definieren der speziellen LOR-Schlüssel
42     spezielle_keys = [999900, 999999]
43
44     # Trennen der regulären und speziellen Zeilen
45     regulär_df = akkumuliert_df[~akkumuliert_df['LOR-Schlüssel'].isin(
46         spezielle_keys)]
47     spezielle_df = akkumuliert_df[akkumuliert_df['LOR-Schlüssel'].isin(
48         spezielle_keys)]
49
50     # Sortieren der regulären Zeilen nach 'Straftaten_insgesamt' absteigend
51     sortiert_regulär_df = regulär_df.sort_values(by='Straftaten_insgesamt',
52         ascending=False)
53
54     # Kombinieren der sortierten regulären Zeilen mit den speziellen Zeilen
55     endgültiges_df = pd.concat([sortiert_regulär_df, spezielle_df],
56         ignore_index=True)
57
58     # 4. Sortiertes DataFrame zurückgeben (hier als Ausgabe angezeigt)
59     print(endgültiges_df)
60
61 if __name__ == '__main__':
62     main()

```

Listing 28: 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
    Spalten summieren
32        group_columns = ['LOR-Schlüssel', 'Bezirke']
33        df_grouped = df_all.groupby(group_columns, as_index=False).sum()
34    except KeyError as e:
35        print(f"Erforderliche Spalte fehlt: {e}")
36        return None
37    except Exception as e:
38        print(f"Fehler beim Zusammenführen der DataFrames: {e}")
39        return None
40
41    # Schritt 3: Sortieren nach 'Straftaten_insgesamt', spezielle LOR-Schlü
    ssel am Ende
42    try:
43        # Definieren der speziellen LOR-Schlüssel
44        special_keys = [999900, 999999]
45
46        # Separieren der speziellen und normalen Zeilen
47        df_special = df_grouped[df_grouped['LOR-Schlüssel'].isin(special_keys
    )]
48        df_normal = df_grouped[~df_grouped['LOR-Schlüssel'].isin(special_keys
    )]
49
50        # Sortieren der normalen Zeilen nach 'Straftaten_insgesamt'

```

```

absteigend
51     df_normal_sorted = df_normal.sort_values(by='Straftaten_insgesamt',
52     ascending=False)
53     # Zusammenführen der sortierten normalen Zeilen mit den speziellen
54     # Zeilen
55     df_final = pd.concat([df_normal_sorted, df_special], ignore_index=
56     True)
57     except KeyError:
58         print("Die Spalte 'Straftaten_insgesamt' ist im DataFrame nicht
59         vorhanden.")
60         return None
61     except Exception as e:
62         print(f"Fehler beim Sortieren des DataFrames: {e}")
63         return None
64
65     # Schritt 4: Rückgabe des sortierten DataFrames
66     return df_final
67
68 if __name__ == "__main__":
69     # Verarbeiten der Excel-Datei und Erhalten des sortierten DataFrames
70     sorted_df = process_fallzahlen_excel('Fallzahlen.xlsx')
71
72     if sorted_df is not None:
73         # Anzeige der ersten paar Zeilen des sortierten DataFrames
74         print(sorted_df.head())
75
76         # Optional: Speichern des sortierten DataFrames in einer neuen Excel-
77         # Datei
78         # sorted_df.to_excel('Fallzahlen_sortiert.xlsx', index=False)

```

Listing 29: 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]

```

```

18
19 # DataFrame ohne die auszuschließenden LOR-Schlüssel
20 zu_sortieren_df = gruppiert_df[~gruppiert_df['LOR-Schlüssel'].isin(
ausschluss_keys)]
21
22 # DataFrame mit den auszuschließenden LOR-Schlüsseln
23 ausgeschlossen_df = gruppiert_df[gruppiert_df['LOR-Schlüssel'].isin(
ausschluss_keys)]
24
25 # Sortieren nach 'Straftaten_insgesamt' in absteigender Reihenfolge
26 sortiert_df = zu_sortieren_df.sort_values(by='Straftaten_insgesamt',
ascending=False)
27
28 # Ausschließen Zeilen am Ende des sortierten DataFrames anhängen
29 final_df = pd.concat([sortiert_df, ausgeschlossen_df], ignore_index=True)
30
31 # 4. Sortiertes DataFrame zurückgeben (hier wird es angezeigt)
32 print(final_df)
33
34 if __name__ == '__main__':
35     main()

```

Listing 30: 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

```

```

26     ().reset_index()
27
28     # 3. Das neue gejointe DataFrame nach der Spalte "Straftaten_insgesamt"
29     sortieren.
30     # Zeilen mit den LOR-Schlüsseln 999900 und 999999 werden nicht in die
31     Sortierung einbezogen
32     # und am Ende des DataFrames angehängt.
33
34     # Trenne die speziellen LOR-Schlüssel
35     special_keys = [999900, 999999]
36     normal_df = joined_df[~joined_df['LOR-Schlüssel'].isin(special_keys)]
37     special_df = joined_df[joined_df['LOR-Schlüssel'].isin(special_keys)]
38
39     # Sortiere das normale DataFrame nach 'Straftaten_insgesamt' absteigend
40     sorted_normal_df = normal_df.sort_values(by='Straftaten_insgesamt',
41     ascending=False)
42
43     # Füge die speziellen Zeilen am Ende hinzu
44     final_df = pd.concat([sorted_normal_df, special_df], ignore_index=True)
45
46     # 4. Das sortierte Pandas DataFrame zurückgeben.
47     return final_df
48
49 if __name__ == "__main__":
50     # Pfad zur Excel-Datei
51     excel_file = 'Fallzahlen.xlsx'
52
53     # Verarbeite die Excel-Datei
54     sorted_df = process_fallzahlen(excel_file)
55
56     # Optional: Zeige die ersten Zeilen des sortierten DataFrames an
57     print(sorted_df.head())
58
59     # Optional: Speichere das sortierte DataFrame in eine neue Excel-Datei
60     sorted_df.to_excel('Fallzahlen_sortiert.xlsx', index=False)

```

Listing 31: 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    """

```



```

14 # Schritt 1: Alle Sheets der Excel-Datei einlesen
15 try:
16     excel_sheets = pd.read_excel(file_path, sheet_name=None)
17 except FileNotFoundError:
18     print(f"Datei '{file_path}' wurde nicht gefunden.")
19     return
20 except Exception as e:
21     print(f"Fehler beim Einlesen der Datei: {e}")
22     return
23
24 # Liste zur Speicherung der DataFrames
25 df_list = []
26
27 for sheet_name, df in excel_sheets.items():
28     # Sicherstellen, dass 'LOR-Schlüssel' und 'Bezirke' als Schlüssel
29     # vorhanden sind
30     if 'LOR-Schlüssel' not in df.columns or 'Bezirke' not in df.columns:
31         print(f"Sheet '{sheet_name}' enthält nicht die notwendigen
32         Spalten.")
33         continue
34     df_list.append(df)
35
36 if not df_list:
37     print("Keine gültigen Sheets gefunden.")
38     return
39
40 # Schritt 2: DataFrames zusammenführen (joinen) auf 'LOR-Schlüssel' und '
41 # Bezirke'
42 merged_df = df_list[0]
43 for df in df_list[1:]:
44     merged_df = pd.merge(merged_df, df, on=['LOR-Schlüssel', 'Bezirke'],
45     how='outer', suffixes=('', '_dup'))
46
47     # Entfernen von doppelten Spalten, falls vorhanden
48     dup_columns = [col for col in merged_df.columns if col.endswith('_dup
49     ')]
50     if dup_columns:
51         merged_df.drop(columns=dup_columns, inplace=True)
52
53 # Schritt 3: Sortieren nach 'Straftaten_insgesamt', wobei 999900 und
54 # 999999 am Ende stehen
55 # Zuerst sicherstellen, dass 'Straftaten_insgesamt' numerisch ist
56 merged_df['Straftaten_insgesamt'] = pd.to_numeric(merged_df['
57 Straftaten_insgesamt'], errors='coerce')
58
59 # Trennen der DataFrames
60 districts_df = merged_df[~merged_df['LOR-Schlüssel'].isin([999900,
61 999999])]
62 non_districts_df = merged_df[merged_df['LOR-Schlüssel'].isin([999900,
63 999999])]
64
65 # Sortieren der Bezirke

```

```

57     sorted_districts_df = districts_df.sort_values(by='Straftaten_insgesamt',
58         ascending=False)
59
60     # Kombinieren der sortierten Bezirke mit den nicht zuordenbaren Einträgen
61     final_df = pd.concat([sorted_districts_df, non_districts_df],
62         ignore_index=True)
63
64     return final_df
65
66 if __name__ == "__main__":
67     # Pfad zur Excel-Datei
68     excel_file = 'Fallzahlen.xlsx'
69
70     # Verarbeitung der Excel-Datei
71     sorted_df = process_fallzahlen_excel(excel_file)
72
73     if sorted_df is not None:
74         # Ausgabe des sortierten DataFrames
75         print(sorted_df)
76
77         # Optional: Speichern des Ergebnisses in einer neuen Excel-Datei
78         # sorted_df.to_excel('Sorted_Fallzahlen.xlsx', index=False)

```

Listing 32: Skript Testfall 2 Prompt 3 Ausführung 5

11.3.3 Testfall 3

```

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'. Erstelle mir ein
19 Python Skript, welches den prozentualen Anteil der gesamten Straftaten der
20 einzelnen Bezirke von den gesamten Straftaten von ganz Berlin berechnet.
21 Hier ist der Inhalt des Sheets 'Fallzahlen_2023': {df}
22 """
23
24 # Anfrage an die OpenAI Chat-API
25 response = client.chat.completions.create(model="o1-mini",

```

```

23 messages=[
24 {"role": "user", "content": prompt}
25 ],)
26
27 response_file = path+"testcases/testcase3/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 33: OpenAI API Request Skript

```

1 import pandas as pd
2
3 def berechne_straftaten_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_straftaten = 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_straftaten) * 100
38
39    # Optional: Runden auf zwei Dezimalstellen

```

```

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

```

Listing 35: 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 36: 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    # Gesamtstrafaten 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 37: 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.")

```



```

45
46     except FileNotFoundError:
47         print("Fehler: Die Datei 'Fallzahlen.xlsx' wurde nicht gefunden.
         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 38: 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']
        ] / gesamt_straftaten) * 100
31
32 # Optional: Runde die Prozentwerte auf zwei Dezimalstellen
33 bezirk_df['Prozent_Straftaten_insgesamt'] = bezirk_df['
        Prozent_Straftaten_insgesamt'].round(2)
34
35 # Ausgabe des Ergebnisses
36 print(bezirk_df[['Bezirke', 'Straftaten_insgesamt', '

```

```

        Prozent_Straftaten_insgesamt']])
37
38 # Optional: Speichern in einer neuen Excel-Datei
39 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 39: 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'] /
    gesamt_straftaten) * 100
45
46 # 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 40: 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'] /
38     gesamt_straftaten) * 100
39
40 # Optional: Runden auf zwei Dezimalstellen
41 bezirke_df['Prozentualer_Anteil'] = bezirke_df['Prozentualer_Anteil'].round
42     (2)
43
44 # Anzeigen der Ergebnisse
45 print(bezirke_df[['Bezirke', 'Straftaten_insgesamt', 'Prozentualer_Anteil']])
46
47 # Speichern der Ergebnisse in einer neuen Excel-Datei
48 ausgabe_datei = 'Fallzahlen_Prozente.xlsx'
49 try:
50     bezirke_df.to_excel(ausgabe_datei, sheet_name='Prozentuale_Fallzahlen',
51         index=False)
52     print(f"\nDie prozentualen Anteile wurden in '{ausgabe_datei}'
53         gespeichert.")
54 except Exception as e:
55     print(f"Ein Fehler ist aufgetreten beim Speichern der Excel-Datei: {e}")

```

Listing 41: 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_gesamtzeile = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
12 if berlin_gesamtzeile.empty:
13     raise ValueError("Die Zeile 'Berlin (PKS gesamt)' wurde in den Daten
14         nicht gefunden.")
15
16 berlin_gesamt = berlin_gesamtzeile['Straftaten_insgesamt'].iloc[0]
17
18 # Entferne die Gesamtzeile aus dem DataFrame
19 bezirke_df = df[df['Bezirke'] != 'Berlin (PKS gesamt)'].copy()
20
21 # 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 42: 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 43: 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
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
31    else:
32        print("Alle erforderlichen Spalten sind vorhanden.")
33
34    # 4. Gesamtzahl der Straftaten für ganz Berlin finden
35    gesamt_row = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
36
37    if gesamt_row.empty:
38        print("Fehler: Die Zeile mit 'Berlin (PKS gesamt)' wurde nicht
39            gefunden.")
40        return
41
42    gesamt_straftaten = gesamt_row['Straftaten_insgesamt'].values[0]
43    print(f"Gesamtzahl der Straftaten für Berlin: {gesamt_straftaten}")
44
45    # 5. Prozentualen Anteil der einzelnen Bezirke berechnen
46    # Ausschließen der Gesamtzeile für die Berechnung
47    df_bezirke = df[df['Bezirke'] != 'Berlin (PKS gesamt)'].copy()
48
49    if df_bezirke.empty:
50        print("Fehler: Es gibt keine Bezirke nach dem Ausschluss von 'Berlin
51            (PKS gesamt)'.")
52        return
53
54    # Prozentualer Anteil berechnen
55    df_bezirke['Prozentualer_Anteil (%)'] = (df_bezirke['Straftaten_insgesamt']
56        / gesamt_straftaten) * 100
57
58    # Optional: Runden auf zwei Dezimalstellen
59    df_bezirke['Prozentualer_Anteil (%)'] = df_bezirke['Prozentualer_Anteil

```

```

(%)'].round(2)

54
55 print("Prozentualer Anteil der Straftaten pro Bezirk berechnet.")
56
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 44: 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 prozentualem
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
26 not in df.columns]
27 if fehlende_spalten:
28     raise ValueError(f"Die folgenden erforderlichen Spalten fehlen in der
29     Tabelle: {fehlende_spalten}")
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 if gesamt_row.empty:
36     raise ValueError("Die Zeile mit 'Berlin (PKS gesamt)' wurde nicht
37     gefunden.")
38
39 gesamt_straftaten = gesamt_row['Straftaten_insgesamt'].iloc[0]
40 print(f"Gesamtzahl der Straftaten in Berlin: {gesamt_straftaten}")
41
42 if gesamt_straftaten == 0:
43     raise ValueError("Die Gesamtzahl der Straftaten ist 0, kann keine
44     Prozente berechnen.")
45
46 # 5. Prozentualen Anteil der einzelnen Bezirke berechnen
47 # Ausschließen der Gesamtzeile von den einzelnen Bezirken
48 einzel_bezirke = df[df['Bezirke'] != 'Berlin (PKS gesamt)'].copy()
49
50 # Berechnung des prozentualen Anteils
51 einzel_bezirke['Prozentualer_Anteil'] = (einzel_bezirke['
52 Straftaten_insgesamt'] / gesamt_straftaten) * 100
53
54 # Optional: Auf zwei Dezimalstellen runden
55 einzel_bezirke['Prozentualer_Anteil'] = einzel_bezirke['
56 Prozentualer_Anteil'].round(2)
57
58 # 6. Ergebnis als DataFrame zurückgeben
59 ergebnis = einzel_bezirke[['Bezirke', 'Straftaten_insgesamt', '
60 Prozentualer_Anteil']]
61
62 return ergebnis
63
64 if __name__ == "__main__":
65     dateipfad = 'Fallzahlen.xlsx' # Pfad zur Excel-Datei anpassen, falls
66     erforderlich
67     try:
68         ergebnis_df = verarbeite_fallzahlen(dateipfad)
69         print("\nProzentualer Anteil der Straftaten pro Bezirk in Berlin:")
70         print(ergebnis_df.to_string(index=False))
71     except Exception as e:
72         print(f"Ein Fehler ist aufgetreten: {e}")

```

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

```

```

total_straftaten) * 100
47
48 # Optional: Runden auf zwei Dezimalstellen
49 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 46: 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
13        return
14
15    # 2. Die Tabelle als Pandas DataFrame gespeichert (bereits in 'df')
16
17    # 3. Überprüfen, ob die notwendigen Spalten vorhanden sind
18    notwendige_spalten = ['Bezirke', 'Straftaten_insgesamt']
19    fehlende_spalten = [spalte for spalte in notwendige_spalten wenn spalte
nicht in df.columns]
20    wenn fehlende_spalten:
21        print(f"Die folgenden notwendigen Spalten fehlen: {'', '.join(
fehlende_spalten)}")
22        return
23
24    # 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 47: 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
Tabelle: {fehlende_spalten}")
32         return
33     else:
34         print("Überprüfung der notwendigen Spalten erfolgreich.")
35
36     # 4. Gesamtzahl der Straftaten für ganz Berlin finden
37     gesamt_row = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
38     if gesamt_row.empty:
39         print("Fehler: Keine Zeile mit 'Bezirke' als 'Berlin (PKS gesamt)'
gefunden.")
40         return
41
42     gesamt_straftaten = gesamt_row['Straftaten_insgesamt'].iloc[0]
43     print(f"Gesamtzahl der Straftaten in Berlin: {gesamt_straftaten}")
44
45     # 5. Prozentualen Anteil der einzelnen Bezirke berechnen
46     # Ausschließen der Gesamtzeile aus den Bezirken
47     bezirke_df = df[df['Bezirke'] != 'Berlin (PKS gesamt)'].copy()
48
49     # Berechnung des prozentualen Anteils
50     bezirke_df['Prozentualer_Anteil'] = (bezirke_df['Straftaten_insgesamt'] /
gesamt_straftaten) * 100
51
52     # Optional: Auf zwei Dezimalstellen runden
53     bezirke_df['Prozentualer_Anteil'] = bezirke_df['Prozentualer_Anteil'].
round(2)
54
55     # 6. Ergebnis als DataFrame zurückgeben
56     ergebnis_df = bezirke_df[['Bezirke', 'Straftaten_insgesamt', '
Prozentualer_Anteil']]
57
58     return ergebnis_df
59
60 # Beispielhafte Nutzung des Skripts
61 if __name__ == "__main__":
62     dateipfad = 'Fallzahlen.xlsx' # Pfad zur Excel-Datei
63     ergebnis = berechne_prozentanteile(dateipfad)
64
65     if ergebnis is not None:

```

```

66     print("\nProzentuale Anteile der Straftaten pro Bezirk:")
67     print(ergebnis)
68
69     # Optional: Ergebnis in eine neue Excel-Datei speichern
70     # ergebnis.to_excel('Prozentuale_Anteile_Straftaten.xlsx', index=
False)

```

Listing 48: Skript Testfall 3 Prompt 3 Ausführung 5

11.3.4 Testfall 4

```

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 df = pd.read_excel(file, sheet_name='Fallzahlen_2023')
14
15 # Prompt erstellen: CSV-Daten einbetten + Anweisung
16 prompt = f"""
17 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.
18 Hier sind die Daten eines der Sheets als Beispiel: {df}
19 """
20
21 # Anfrage an die OpenAI Chat-API
22 response = client.chat.completions.create(model="o1-mini",
23 messages=[
24 {"role": "user", "content": prompt}
25 ],)
26
27 response_file = path+"testcases/testcase4/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 49: OpenAI API Request Skript

```

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 = []
46
47     for sheet in sheet_names:
48         try:
49             # Lade das Sheet als DataFrame
50             df = pd.read_excel(xls, sheet_name=sheet)
51
52             # Extrahiere das Jahr
53             year = extract_year(sheet)

```

```

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 50: 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
8     Bezirksnamen
9     'Berlin (PKS gesamt)' gesucht.
10
11     Args:
12         df (pd.DataFrame): Der DataFrame des aktuellen Sheets.
13
14     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
62                     Jahresformat und wird übersprungen.")
63                 continue
64
65             # 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']
86     .pct_change() * 100
87
88     # Optional: Runde die Prozentwerte auf zwei Dezimalstellen
89     df_totals['Prozentuale_Veraenderung'] = df_totals['
90     Prozentuale_Veraenderung'].round(2)
91
92     # Anzeige des resultierenden DataFrames
93     print(df_totals)
94
95     # Optional: Speichere das Ergebnis in eine neue Excel-Datei
96     output_file = 'Straftaten_Zeitreihe.xlsx'
97     try:
98         df_totals.to_excel(output_file)
99         print(f"Das Ergebnis wurde erfolgreich in '{output_file}' gespeichert
100         .")
101     except Exception as e:
102         print(f"Fehler beim Speichern der Ergebnisdatei: {e}")
103
104 if __name__ == "__main__":
105     main()

```

Listing 51: 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 52: 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 Gesamtstrafataten
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
        vorhanden sind
16     if 'LOR-Schlüssel' in df.columns and 'Straftaten_insgesamt' in df.columns
        :
17         # Suche die Zeile, die die Gesamtzahl für Berlin enthält
18         # Annahme: 'LOR-Schlüssel' 999999 oder 'Berlin (PKS gesamt)' reprä
        sentiert die Gesamtdaten
19         total_row = df[df['LOR-Schlüssel'] == 999999]
20
21         # Falls nicht gefunden, versuche mit dem Namen
22         if total_row.empty:
23             total_row = df[df['LOR-Schlüssel'].astype(str).str.contains('
Berlin \ (PKS gesamt\)', regex=True)]
24
25         # Falls immer noch nicht gefunden, überspringe das Sheet
26         if total_row.empty:
27             print(f"Warnung: Gesamtdaten nicht in Sheet '{sheet_name}'
        gefunden.")
28             continue
29
30         # Extrahiere das Jahr aus dem Sheet-Namen
31         # Annahme: Der Sheet-Name enthält das Jahr, z.B. "2020", "Jahr 2020",
        etc.
32         # Hier wird versucht, eine vierstellige Zahl zu extrahieren
33         import re
34         match = re.search(r'\b(19|20)\d{2}\b', sheet_name)
35         if match:
36             year = int(match.group())
37         else:
38             print(f"Warnung: Jahr konnte nicht aus dem Sheet-Namen '{
        sheet_name}' extrahiert werden. Überspringe dieses Sheet.")
39             continue
40
41         # Extrahiere die Gesamtzahl der Straftaten
42         total_crimes = total_row['Straftaten_insgesamt'].values[0]
43
44         # Füge die Daten zur Liste hinzu
45         data.append({'Jahr': year, 'Straftaten_insgesamt': total_crimes})
46     else:
47         print(f"Warnung: Erforderliche Spalten nicht in Sheet '{sheet_name}'
        vorhanden.")
48
49 # Erstelle einen DataFrame aus den gesammelten Daten
50 df_total = pd.DataFrame(data)
51
52 # Sortiere den DataFrame nach dem Jahr
53 df_total = df_total.sort_values('Jahr').reset_index(drop=True)
54
55 # Berechne die prozentuale Veränderung zum Vorjahr

```

```

56 df_total['Prozentuale_Veraenderung_zum_Vorjahr (%)'] = df_total['
    Straftaten_insgesamt'].pct_change() * 100
57
58 # Optional: Rundung der prozentualen Veränderung auf zwei Dezimalstellen
59 df_total['Prozentuale_Veraenderung_zum_Vorjahr (%)'] = df_total['
    Prozentuale_Veraenderung_zum_Vorjahr (%)'].round(2)
60
61 # Setze das Jahr als Index (optional)
62 df_total.set_index('Jahr', inplace=True)
63
64 # Anzeige des Ergebnis-DataFrames
65 print(df_total)
66
67 # Optional: Speichere das Ergebnis in eine neue Excel-Datei
68 df_total.to_excel('Zeitreihe_Straftaten_Berlin.xlsx')

```

Listing 53: 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 for sheet in xls.sheet_names:
14     # Daten des aktuellen Sheets lesen
15     df = pd.read_excel(xls, sheet_name=sheet)
16
17     # Sicherstellen, dass die 'LOR-Schlüssel' Spalte als String behandelt
    wird
18     df['LOR-Schlüssel'] = df['LOR-Schlüssel'].astype(str)
19
20     # Filtern nach der Zeile für ganz Berlin (angenommen '999999' ist der
    Schlüssel)
21     berlin_gesamt = df[df['LOR-Schlüssel'] == '999999']
22
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
    alle Bezirke
28         straftaten = df['Straftaten_insgesamt'].sum()
29
30     # Füge die Daten zur Liste hinzu
31     jahresdaten.append({'Jahr': sheet, 'Straftaten_insgesamt': straftaten})

```

```

32
33 # Erstelle einen DataFrame aus den gesammelten Daten
34 df_gesamt = pd.DataFrame(jahresdaten)
35
36 # Versuche, die 'Jahr' Spalte in Integer zu konvertieren für die Sortierung
37 try:
38     df_gesamt['Jahr'] = df_gesamt['Jahr'].astype(int)
39     df_gesamt = df_gesamt.sort_values('Jahr')
40 except ValueError:
41     # Falls die Sheet-Namen nicht numerisch sind, sortiere alphabetisch
42     df_gesamt = df_gesamt.sort_values('Jahr')
43
44 # Setze das Jahr als Index
45 df_gesamt.set_index('Jahr', inplace=True)
46
47 # Berechne die prozentuale Veränderung zum Vorjahr
48 df_gesamt['% Veränderung Vorjahr'] = df_gesamt['Straftaten_insgesamt'].
    pct_change() * 100
49
50 # Runde die Prozentwerte auf zwei Dezimalstellen
51 df_gesamt['% Veränderung Vorjahr'] = df_gesamt['% Veränderung Vorjahr'].round
    (2)
52
53 # Anzeige des resultierenden DataFrames
54 print(df_gesamt)
55
56 # Optional: Speichern des DataFrames in eine neue Excel-Datei
57 df_gesamt.to_excel('Prozentuale_Veraenderung_Straftaten_Berlin.xlsx')

```

Listing 54: 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
    bersprungen.")
19         continue
20
21 # Sortiere die Sheets nach Jahr

```

```

22 sheet_years_sorted = sorted(sheet_years, key=lambda x: x[0])
23
24 # Liste zur Speicherung der Ergebnisse
25 data = []
26
27 # Iteriere über jedes Sheet und extrahiere die benötigten Daten
28 for year, sheet in sheet_years_sorted:
29     # Lese das aktuelle Sheet
30     df = pd.read_excel(excel_file, sheet_name=sheet)
31
32     # Stelle sicher, dass die benötigten Spalten vorhanden sind
33     if 'Bezirke' not in df.columns or 'Straftaten_insgesamt' not in df.
        columns:
34         print(f"Sheet '{sheet}' fehlt die erforderliche Spalte. Überspringe
            dieses Sheet.")
35         continue
36
37     # Filtere die Zeile für "Berlin (PKS gesamt)"
38     row = df[df['Bezirke'] == "Berlin (PKS gesamt)"]
39
40     if not row.empty:
41         # Extrahiere den Wert und stelle sicher, dass er numerisch ist
42         value = row['Straftaten_insgesamt'].values[0]
43         if pd.notnull(value):
44             data.append({'Year': year, 'Straftaten_insgesamt': value})
45         else:
46             print(f"Der Wert für 'Straftaten_insgesamt' im Jahr {year} ist
                NaN.")
47         else:
48             print(f"'Berlin (PKS gesamt)' nicht in Sheet '{sheet}' gefunden.")
49
50 # Erstelle ein DataFrame aus den gesammelten Daten
51 df_data = pd.DataFrame(data)
52
53 # Sortiere das DataFrame nach Jahr, um sicherzustellen, dass die Reihenfolge
    stimmt
54 df_data = df_data.sort_values('Year').reset_index(drop=True)
55
56 # Berechne die prozentuale Veränderung zum Vorjahr
57 df_data['Prozentuale_Veraenderung'] = df_data['Straftaten_insgesamt'].
    pct_change() * 100
58
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 55: Skript Testfall 4 Prompt 2 Ausführung 1

```
1 import pandas as pd
2
3 def compute_percentage_change(excel_file, target_row, target_column,
4                               sheet_prefix):
5     """
6     Liest eine Excel-Datei mit mehreren Sheets, extrahiert die
7     Straftaten_insgesamt für
8     einen bestimmten Eintrag und berechnet die prozentuale Veränderung zum
9     Vorjahr.
10
11     Parameters:
12     - excel_file: str, Pfad zur Excel-Datei.
13     - target_row: str, Der Eintrag in der 'Bezirke' Spalte, z.B. "Berlin (PKS
14       gesamt)".
15     - target_column: str, Name der Spalte, deren prozentuale Veränderung
16       berechnet werden soll, z.B. "Straftaten_insgesamt".
17     - sheet_prefix: str, Präfix der Sheet-Namen, z.B. "Fallzahlen_" für
18       Sheets wie "Fallzahlen_2014".
19
20     Returns:
21     - pandas.DataFrame mit den Jahren und der prozentualen Veränderung.
22     """
23     try:
24         # Alle Sheets einlesen
25         sheets_dict = pd.read_excel(excel_file, sheet_name=None, engine='
26         openpyxl')
27     except FileNotFoundError:
28         print(f"Die Datei {excel_file} wurde nicht gefunden.")
29         return None
30     except Exception as e:
31         print(f"Ein Fehler ist aufgetreten beim Einlesen der Datei: {e}")
32         return None
33
34     data = []
35
36     for sheet_name, df in sheets_dict.items():
37         # Überprüfen, ob der Sheet-Name dem erwarteten Muster entspricht
38         if not sheet_name.startswith(sheet_prefix):
39             print(f"Überspringe Sheet '{sheet_name}', da es nicht mit '{
40             sheet_prefix}' beginnt.")
41             continue
42
43         # Extrahiere das Jahr aus dem Sheet-Namen
44         try:
45             year_str = sheet_name.replace(sheet_prefix, "")
46             year = int(year_str)
47         except ValueError:
48             print(f"Konnte das Jahr aus dem Sheet-Namen '{sheet_name}' nicht
49             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,
target_column, sheet_prefix)
82
83     if percentage_change_df is not None:
84         print("Prozentuale Veränderung der Straftaten_insgesamt zum Vorjahr:"
)
85         print(percentage_change_df)
86
87     # Optional: DataFrame speichern

```



```

88         # percentage_change_df.to_csv('prozentuale_veraenderung.csv', index=
           False)

```

Listing 56: 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
40
41 # Erstelle einen DataFrame aus dem Dictionary
42 df_crimes = pd.DataFrame(list(data.items()), columns=['Jahr', '
43     Straftaten_insgesamt'])
44
45 # Sortiere den DataFrame nach Jahr
46 df_crimes = df_crimes.sort_values('Jahr').reset_index(drop=True)
47
48 # 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 57: 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 58: 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:
26         return row.iloc[0]['Straftaten_insgesamt']
27     else:
28         print(f"Warnung: Bezirk '{target_district}' nicht in Sheet '{
29         sheet_name}' gefunden.")
30         return None
31
32 def main():
33     excel_file = 'Fallzahlen.xlsx'
34     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 )
50     return
51
52 # Sortiere die Sheets nach Jahr
53 sheets_with_year.sort(key=lambda x: x[0])
54
55 # Extrahiere die 'Straftaten_insgesamt' Werte
56 data = {}
57 for year, sheet in sheets_with_year:
58     value = get_straftaten_insgesamt(excel_file, sheet, target_district)
59     if value is not None:
60         data[year] = value
61
62 # Erstelle ein DataFrame
63 df = pd.DataFrame(list(data.items()), columns=['Jahr', 'Straftaten_insgesamt'])
64 df.sort_values('Jahr', inplace=True)
65 df.reset_index(drop=True, inplace=True)
66
67 # Berechne die prozentuale Veränderung zum Vorjahr
68 df['Prozentuale_Veraenderung'] = df['Straftaten_insgesamt'].pct_change()
69 * 100
70
71 # Optional: Runde die Prozentwerte auf zwei Dezimalstellen
72 df['Prozentuale_Veraenderung'] = df['Prozentuale_Veraenderung'].round(2)
73
74 print(df)
75
76 if __name__ == "__main__":
77     main()

```

Listing 59: 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 strafaten_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         strafaten_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(strafaten_gesamt.keys())
45 prozentuale_veraenderung = {}
46
47 for i in range(1, len(jahre)):
48     aktuelles_jahr = jahre[i]
49     vorheriges_jahr = jahre[i-1]
50     wert_aktuell = strafaten_gesamt[aktuelles_jahr]
51     wert_vorher = strafaten_gesamt[vorheriges_jahr]
52     veraenderung = ((wert_aktuell - wert_vorher) / wert_vorher) * 100
53     prozentuale_veraenderung[aktuelles_jahr] = veraenderung
54
55 # Schritt 4: Ergebnisse in einem neuen DataFrame speichern
56 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 60: 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 61: 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 '
23         Fallzahlen_YYYY')
24         try:
25             year = int(sheet.split('_')[-1])
26         except ValueError:
27             print(f"Sheet-Name {sheet} entspricht nicht dem erwarteten Format
28             'Fallzahlen_YYYY'.")
29             continue
30
31         # Finde die Zeile für 'Berlin (PKS gesamt)'
32         berlin_row = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
33
34         if not berlin_row.empty:
35             # Extrahiere den Wert von 'Straftaten_insgesamt'
36             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 62: 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():
13     # Extrahiere das Jahr aus dem Sheet-Namen, z.B. 'Fallzahlen_2014' -> 2014
14     try:
15         year = int(sheet_name.split('_')[-1])
16     except ValueError:
17         print(f"Sheet-Name '{sheet_name}' entspricht nicht dem erwarteten
18   Muster 'Fallzahlen_Jahr'. Übersprungen.")
19         continue
20
21 # 2. Extrahiere den Wert der Spalte 'Straftaten_insgesamt' für die Zeile

```

```

19     'Berlin (PKS gesamt)'
20     # Überprüfen Sie, ob die Spaltennamen korrekt sind. Möglicherweise müssen
    Sie 'Bezirke' anpassen.
21     # Hier wird angenommen, dass die Spalte mit Bezirken 'Bezirke' heißt
    berlin_row = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
22
23     if not berlin_row.empty:
24         straf_value = berlin_row['Straftaten_insgesamt'].values[0]
25         straftaten_dict[year] = straf_value
26     else:
27         print(f"'Berlin (PKS gesamt)' nicht in Sheet '{sheet_name}' gefunden.
    ")
28
29 # 3. Berechnen Sie die prozentuale Veränderung zum jeweiligen Vorjahr
30 # Sortieren der Jahre
31 sorted_years = sorted(straftaten_dict.keys())
32 percentage_changes = []
33
34 for i in range(1, len(sorted_years)):
35     current_year = sorted_years[i]
36     previous_year = sorted_years[i - 1]
37     current_value = straftaten_dict[current_year]
38     previous_value = straftaten_dict[previous_year]
39
40     # Prozentuale Veränderung berechnen
41     if previous_value != 0:
42         percent_change = ((current_value - previous_value) / previous_value)
    * 100
43     else:
44         percent_change = None # Vermeiden von Division durch Null
45
46     percentage_changes.append({
47         'Jahr': current_year,
48         'Prozentuale Veränderung': percent_change
49     })
50
51 # 4. Speichern Sie die Ergebnisse in einem neuen DataFrame
52 result_df = pd.DataFrame(percentage_changes)
53
54 # Optional: Hinzufügen des ersten Jahres mit einer NaN für die Veränderung
55 result_df = pd.concat([
56     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 63: 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 64: Skript Testfall 4 Prompt 3 Ausführung 5

11.3.5 Testfall 5

```

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&HZ 2014-2023.xlsx"
13 df = pd.read_excel(file, sheet_name='Fallzahlen_2023')
14
15 # Prompt erstellen: CSV-Daten einbetten + Anweisung
16 prompt = f"""
17 Ich habe eine Excel Datei mit dem Namen 'Fallzahlen&HZ 2014-2023.xlsx'.
18     Erstelle mir ein Python Skript, das die Daten des Sheets 'Fallzahlen_2023'
19     ausliest und für jeden Oberbezirk den Unterbezirk mit den meisten
20     Raubdelikten ausgibt.
21 Hier sind die Daten des Sheets zur Orientierung: {df}
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 ],)
29
30 response_file = path+"testcases/testcase5/prompt1/exec5/response.txt"
31 os.makedirs(os.path.dirname(response_file), exist_ok=True)
32 # Ausgabe des generierten Skripts
33 with open(response_file, "w") as file:
34     file.write(response.choices[0].message.content)
35
36 print(f"Response wurde in {response_file} gespeichert.")

```

Listing 65: OpenAI API Request Skript

```
1 import pandas as pd
2
3 # Liste der Oberbezirke in Berlin
4 oberbezirke = [
5     'Mitte',
6     'Friedrichshain-Kreuzberg',
7     'Pankow',
8     'Charlottenburg-Wilmersdorf',
9     'Spandau',
10    'Steglitz-Zehlendorf',
11    'Tempelhof-Schöneberg',
12    'Neukölln',
13    'Treptow-Köpenick',
14    'Marzahn-Hellersdorf',
15    'Lichtenberg',
16    'Reinickendorf'
17 ]
18
19 def get_oberbezirk(bezeichnung):
20     """
21     Bestimmt den Oberbezirk basierend auf der Bezeichnung des Bezirks.
22     """
23     for oberbezirk in oberbezirke:
24         if bezeichnung.startswith(oberbezirk):
25             return oberbezirk
26     return None # Rückgabe von None, wenn kein Oberbezirk gefunden wird
27
28 def main():
29     # Pfad zur Excel-Datei
30     datei_pfad = 'Fallzahlen&HZ 2014-2023.xlsx'
31
32     try:
33         # Einlesen des spezifischen Sheets
34         df = pd.read_excel(datei_pfad, sheet_name='Fallzahlen_2023')
35     except FileNotFoundError:
36         print(f"Die Datei '{datei_pfad}' wurde nicht gefunden.")
37         return
38     except ValueError:
39         print("Das Sheet 'Fallzahlen_2023' existiert nicht in der Excel-Datei")
40         return
41
42     # Überprüfen, ob die notwendigen Spalten vorhanden sind
43     notwendige_spalten = ['Bezeichnung (Bezirksregion)', 'Raub']
44     for spalte in notwendige_spalten:
45         if spalte not in df.columns:
46             print(f"Die Spalte '{spalte}' ist in den Daten nicht vorhanden.")
47             return
48
```

```

49     # Hinzufügen einer neuen Spalte für den Oberbezirk
50     df['Oberbezirk'] = df['Bezeichnung (Bezirksregion)'].apply(get_oberbezirk
51 )
52
53     # Entfernen von Einträgen, die keinem Oberbezirk zugeordnet werden können
54     df = df.dropna(subset=['Oberbezirk'])
55
56     # Gruppieren nach Oberbezirk und Finden des Unterbezirks mit den meisten
57     Raubdelikten
58     top_unterbezirke = df.loc[df.groupby('Oberbezirk')['Raub'].idxmax()]['Oberbezirk', 'Bezeichnung (Bezirksregion)', 'Raub']
59
60     # Ausgabe der Ergebnisse
61     print("Unterbezirk mit den meisten Raubdelikten pro Oberbezirk (2023):\n")
62
63     for index, row in top_unterbezirke.iterrows():
64         oberbezirk = row['Oberbezirk']
65         unterbezirk = row['Bezeichnung (Bezirksregion)']
66         raub = row['Raub']
67         print(f"Oberbezirk: {oberbezirk}")
68         print(f"    Unterbezirk: {unterbezirk}")
69         print(f"    Anzahl Raubdelikte: {raub}\n")
70
71 if __name__ == "__main__":
72     main()

```

Listing 66: Skript Testfall 5 Prompt 1 Ausführung 1

```

1  import pandas as pd
2
3  # Pfad zur Excel-Datei
4  excel_file = 'Fallzahlen&HZ 2014-2023.xlsx'
5
6  # Name des Sheets
7  sheet_name = 'Fallzahlen_2023'
8
9  # Laden des Sheets in einen DataFrame
10 try:
11     df = pd.read_excel(excel_file, sheet_name=sheet_name)
12 except FileNotFoundError:
13     print(f"Die Datei '{excel_file}' wurde nicht gefunden.")
14     exit(1)
15 except ValueError:
16     print(f"Das Sheet '{sheet_name}' existiert nicht in der Datei.")
17     exit(1)
18
19 # Überprüfen der relevanten Spalten
20 required_columns = [
21     'LOR-Schlüssel (Bezirksregion)',
22     'Bezeichnung (Bezirksregion)',
23     'Raub'
24 ]
25

```

```

26 for col in required_columns:
27     if col not in df.columns:
28         print(f"Die erforderliche Spalte '{col}' ist im Sheet nicht vorhanden
29             .")
30         exit(1)
31 # Entfernen von Zeilen, die nicht zugeordnet sind (optional)
32 # Angenommen, LOR-Schlüssel '129900', '999900', '999999' sind nicht
33     zuzuordnen
34 invalid_keys = ['129900', '999900', '999999']
35 df = df[~df['LOR-Schlüssel (Bezirksregion)'].astype(str).str.startswith(tuple
36     (invalid_keys))]
37 # Extrahieren des Oberbezirkscodes (angenommen, die ersten 2 Stellen)
38 df['Oberbezirk_Code'] = df['LOR-Schlüssel (Bezirksregion)'].astype(str).str
39     [:2]
40 # Optional: Mapping der Oberbezirk-Codes zu Namen
41 # Dies muss entsprechend deiner Daten angepasst werden
42 oberbezirk_mapping = {
43     '10': 'Mitte',
44     '11': 'Pankow',
45     '12': 'Charlottenburg-Wilmersdorf',
46     '13': 'Friedrichshain-Kreuzberg',
47     '14': 'Lichtenberg',
48     '15': 'Marzahn-Hellersdorf',
49     '16': 'Treptow-Köpenick',
50     '17': 'Tempelhof-Schöneberg',
51     '18': 'Neukölln',
52     '19': 'Steglitz-Zehlendorf',
53     # Füge weitere Codes und Namen hinzu, falls vorhanden
54 }
55 df['Oberbezirk'] = df['Oberbezirk_Code'].map(oberbezirk_mapping)
56 # Überprüfen auf fehlende Zuordnungen
57 missing_mapping = df[df['Oberbezirk'].isna()]['Oberbezirk_Code'].unique()
58 if len(missing_mapping) > 0:
59     print("Warnung: Einige Oberbezirk-Codes konnten nicht zugeordnet werden:"
60         , missing_mapping)
61     # Optional: Weiterverarbeiten oder Abbruch
62     # exit(1)
63 # Gruppieren nach Oberbezirk und Finden des Unterbezirks mit den meisten
64     Raubdelikten
65 result = df.loc[df.groupby('Oberbezirk')['Raub'].idxmax()][['Oberbezirk', '
66     Bezeichnung (Bezirksregion)', 'Raub']]
67 # Sortieren nach Oberbezirk
68 result = result.sort_values('Oberbezirk')
69 # Ergebnis anzeigen

```

```

71 print("Unterbezirk mit den meisten Raubdelikten pro Oberbezirk:")
72 print(result.to_string(index=False))
73
74 # Optional: Ergebnis in eine neue Excel-Datei speichern
75 output_file = 'Meiste_Raubdelikte_pro_Oberbezirk.xlsx'
76 result.to_excel(output_file, index=False)
77 print(f"\nDas Ergebnis wurde in '{output_file}' gespeichert.")

```

Listing 67: Skript Testfall 5 Prompt 1 Ausführung 2

```

1 import pandas as pd
2
3 # Pfad zur Excel-Datei
4 file_path = 'Fallzahlen&HZ 2014-2023.xlsx'
5 sheet_name = 'Fallzahlen_2023'
6
7 # Excel-Datei einlesen
8 df = pd.read_excel(file_path, sheet_name=sheet_name)
9
10 # Anzeige der ersten Zeilen zur Überprüfung (optional)
11 # print(df.head())
12
13 # Annahme: Der LOR-Schlüssel hat mindestens 5 Stellen
14 # Extrahieren des Oberbezirksskennung (erste zwei Stellen)
15 df['Oberbezirk_Key'] = df['LOR-Schlüssel'].astype(str).str[:2]
16
17 # Extrahieren des Unterbezirksskennung (ganzer Schlüssel)
18 df['Unterbezirk_Key'] = df['LOR-Schlüssel'].astype(str)
19
20 # Gruppieren nach Oberbezirk_Key und Bezeichnung des Oberbezirks
21 # Hier nehmen wir an, dass die Bezeichnung des Oberbezirks dieselbe ist für
    alle Unterbezirke
22 oberbezirke = df.groupby('Oberbezirk_Key')['Bezeichnung'].first().reset_index()
23 oberbezirke = oberbezirke.rename(columns={'Bezeichnung': 'Oberbezirk_Name'})
24
25 # Merge der Oberbezirk Informationen zurück zum Haupt-DataFrame
26 df = df.merge(oberbezirke, on='Oberbezirk_Key', how='left')
27
28 # Gruppieren nach Oberbezirk und Finden des Unterbezirks mit den meisten
    Raubdelikten
29 # Annahme: Die Spalte für Raubdelikte heißt 'Raub'
30 result = df.groupby('Oberbezirk_Name').apply(
31     lambda x: x.loc[x['Raub'].idxmax()][['Bezeichnung', 'Raub']]
32 ).reset_index()
33
34 # Umbenennen der Spalten für Klarheit
35 result = result.rename(columns={
36     'Bezeichnung': 'Unterbezirk_mit_meisten_Raub',
37     'Raub': 'Anzahl_Raub'
38 })
39
40 # Anzeige des Ergebnisses

```



```

41 print(result)
42
43 # Optional: Speichern des Ergebnisses in eine CSV-Datei
44 result.to_csv('Unterbezirk_mit_meisten_Raub_pro_Oberbezirk_2023.csv', index=
    False)

```

Listing 68: Skript Testfall 5 Prompt 1 Ausführung 3

```

1 import pandas as pd
2
3 def finde_oberbezirke_mit_meisten_raub(excel_datei, sheet_name):
4     # Excel-Datei lesen
5     try:
6         df = pd.read_excel(excel_datei, sheet_name=sheet_name, dtype={'LOR-
LOR-Schlüssel': str})
7     except Exception as e:
8         print(f"Fehler beim Lesen der Excel-Datei: {e}")
9         return
10
11     # Sicherstellen, dass 'LOR-Schlüssel' als String gelesen werden und fü
hrende Nullen beibehalten
12     df['LOR-Schlüssel'] = df['LOR-Schlüssel'].str.strip().str.zfill(6)
13
14     # Entfernen von nicht zuzuordenenden Bezirken
15     df = df[~df['Bezeichnung (Bezirksregion)'].str.contains('nicht zuzuordnen
', na=False)]
16
17     # Identifizieren der Oberbezirke: 5-stelliger LOR-Schlüssel, der mit
'000' endet
18     oberbezke = df[df['LOR-Schlüssel'].str.endswith('000') & (df['LOR-Schlü
ssel'].str.len() == 5)]
19
20     if oberbezke.empty:
21         print("Keine Oberbezirke gefunden. Überprüfen Sie die Kriterien zur
Identifikation der Oberbezirke.")
22         return
23
24     # Liste der Oberbezirke
25     oberbezke_list = oberbezke[['LOR-Schlüssel', 'Bezeichnung (Bezirksregion)
']].to_dict('records')
26
27     # Ergebnisse speichern
28     ergebnisse = []
29
30     for ober in oberbezke_list:
31         ober_code = ober['LOR-Schlüssel']
32         ober_name = ober['Bezeichnung (Bezirksregion)']
33
34         # Finden der Unterbezirke, die mit dem Oberbezirkscode beginnen
35         # Annahme: Der Oberbezirkscode ohne die letzten drei Nullen ist der
Präfix
36         prefix = ober_code[:-3] # z.B. '10' aus '10000'
37

```

```

38     # Unterbezirke finden
39     unterbezke = df[
40         (df['LOR-Schlüssel'].str.startswith(prefix)) &
41         (df['LOR-Schlüssel'] != ober_code)
42     ]
43
44     if unterbezke.empty:
45         print(f"Keine Unterbezirke für Oberbezirk {ober_name} gefunden.")
46         continue
47
48     # Unterbezirk mit den meisten Raubdelikten finden
49     max_raub_idx = unterbezke['Raub'].idxmax()
50     max_raub = unterbezke.loc[max_raub_idx]
51     unterbez_name = max_raub['Bezeichnung (Bezirksregion)']
52     raub_count = max_raub['Raub']
53
54     ergebnisse.append({
55         'Oberbezirk': ober_name,
56         'Unterbezirk': unterbez_name,
57         'Raubdelikte': raub_count
58     })
59
60     # Ergebnisse anzeigen
61     print("Unterbezirke mit den meisten Raubdelikten pro Oberbezirk:")
62     for res in ergebnisse:
63         print(f"Oberbezirk: {res['Oberbezirk']}, "
64               f"Unterbezirk: {res['Unterbezirk']}, "
65               f"Raubdelikte: {res['Raubdelikte']}")
66
67 if __name__ == "__main__":
68     excel_datei = 'Fallzahlen&HZ 2014-2023.xlsx' # Pfad zur Excel-Datei
69     sheet_name = 'Fallzahlen_2023' # Name des Sheets
70     finde_oberbezirke_mit_meisten_raub(excel_datei, sheet_name)

```

Listing 69: Skript Testfall 5 Prompt 1 Ausführung 4

```

1 import pandas as pd
2
3 # Pfad zur Excel-Datei
4 excel_datei = 'Fallzahlen&HZ 2014-2023.xlsx'
5
6 # Lesen des Sheets 'Fallzahlen_2023'
7 df = pd.read_excel(excel_datei, sheet_name='Fallzahlen_2023')
8
9 # Anzeigen der ersten Zeilen zur Überprüfung (optional)
10 # print(df.head())
11
12 # Sicherstellen, dass 'LOR-Schlüssel' als String behandelt wird
13 df['LOR-Schlüssel'] = df['LOR-Schlüssel'].astype(str)
14
15 # Extrahieren der Oberbezirk-Information aus den ersten 3 Zeichen des LOR-
    Schlüssels
16 # Passen Sie dies an, falls die Struktur anders ist

```

```

17 df['Oberbezirk_Key'] = df['LOR-Schlüssel'].str[:3]
18
19 # Optional: Definieren Sie eine Mapping-Tabelle für Oberbezirk_Key zu
    Oberbezirk_Name
20 # Hier ist ein Beispiel. Sie müssen dies basierend auf Ihrer spezifischen
    Kodierung anpassen.
21 oberbezirk_mapping = {
22     '100': 'Mitte',
23     '110': 'Cluster A', # Beispielnamen
24     '120': 'Cluster B',
25     # Fügen Sie weitere Zuordnungen hier hinzu
26     # ...
27 }
28
29 # Hinzufügen einer Spalte für den Oberbezirk-Namen
30 df['Oberbezirk'] = df['Oberbezirk_Key'].map(oberbezirk_mapping)
31
32 # Falls bestimmte Keys nicht im Mapping vorhanden sind, können Sie den Key
    als Namen verwenden
33 df['Oberbezirk'] = df['Oberbezirk'].fillna(df['Oberbezirk_Key'])
34
35 # Bereinigen der Daten: Entfernen von Einträgen, die nicht zugeordnet werden
    können
36 ausgeschlossene_terms = ['nicht zuzuordnen', 'gesamt']
37
38 df_clean = df[~df['Bezeichnung (Bezirksregion)'].str.contains('|'.join(
    ausgeschlossene_terms), na=False)]
39
40 # Gruppieren nach Oberbezirk und Finden des Unterbezirks mit den meisten
    Raubdelikten
41 result = df_clean.loc[df_clean.groupby('Oberbezirk')['Raub'].idxmax()][['
    Oberbezirk', 'Bezeichnung (Bezirksregion)', 'Raub']]
42
43 # Sortieren der Ergebnisse nach Oberbezirk (optional)
44 result = result.sort_values('Oberbezirk')
45
46 # Ausgabe der Ergebnisse
47 for index, row in result.iterrows():
48     print(f"Oberbezirk: {row['Oberbezirk']}, Unterbezirk mit den meisten
    Raubdelikten: {row['Bezeichnung (Bezirksregion)']} ({row['Raub']}
    Raubdelikte)")
49
50 # Optional: Speichern der Ergebnisse in einer neuen Excel- oder CSV-Datei
51 # result.to_excel('Ergebnis_Most_Raubdelikte.xlsx', index=False)
52 # oder
53 # result.to_csv('Ergebnis_Most_Raubdelikte.csv', index=False)

```

Listing 70: Skript Testfall 5 Prompt 1 Ausführung 5

```

1 import pandas as pd
2
3 # Pfad zur Excel-Datei
4 excel_datei = 'Fallzahlen&HZ 2014-2023.xlsx'

```

```

5
6 # Name des Sheets
7 sheet_name = 'Fallzahlen_2023'
8
9 # Oberbezirke-Liste
10 oberbezirke = [
11     'Mitte',
12     'Friedrichshain-Kreuzberg',
13     'Pankow',
14     'Charlottenburg-Wilmersdorf',
15     'Spandau',
16     'Steglitz-Zehlendorf',
17     'Tempelhof-Schöneberg',
18     'Neukölln',
19     'Treptow-Köpenick',
20     'Marzahn-Hellersdorf',
21     'Lichtenberg',
22     'Reinickendorf'
23 ]
24
25 # Daten aus dem Excel-Sheet auslesen
26 df = pd.read_excel(excel_datei, sheet_name=sheet_name)
27
28 # Letzte zwei Zeilen (Gesamtwerte) entfernen
29 df = df.iloc[:-2].reset_index(drop=True)
30
31 # Spaltennamen anpassen, falls nötig (Beispiel: Leerzeichen entfernen)
32 df.columns = [col.strip() for col in df.columns]
33
34 # Liste zur Speicherung der Ergebnisse
35 ergebnisse = []
36
37 # Variable zur Speicherung des aktuellen Oberbezirks
38 aktueller_oberbezirk = None
39
40 # Dictionary zur Speicherung der maximalen Raubdelikte pro Oberbezirk
41 max_raub_dict = {}
42
43 for index, row in df.iterrows():
44     bezirk = row['Bezeichnung (Bezirksregion)']
45
46     if bezirk in oberbezirke:
47         # Neuer Oberbezirk gefunden
48         aktueller_oberbezirk = bezirk
49     else:
50         if aktueller_oberbezirk is not None:
51             # Unterbezirk des aktuellen Oberbezirks
52             unterbezirk = bezirk
53             raub = row['Raub']
54
55             # Überprüfen, ob dieser Unterbezirk mehr Raubdelikte hat als
56             vorherige

```

```

56         if aktueller_oberbezirk not in max_raub_dict:
57             max_raub_dict[aktueller_oberbezirk] = {'Unterbezirk':
unterbezirk, 'Raub': raub}
58         else:
59             if raub > max_raub_dict[aktueller_oberbezirk]['Raub']:
60                 max_raub_dict[aktueller_oberbezirk] = {'Unterbezirk':
unterbezirk, 'Raub': raub}
61
62 # Ergebnisse ausgeben
63 print("Unterbezirk mit den meisten Raubdelikten pro Oberbezirk:")
64 for oberbezirk in oberbezirke:
65     if oberbezirk in max_raub_dict:
66         unterbezirk = max_raub_dict[oberbezirk]['Unterbezirk']
67         raub = max_raub_dict[oberbezirk]['Raub']
68         print(f"{oberbezirk}: {unterbezirk} ({raub} Raubdelikte)")
69     else:
70         print(f"{oberbezirk}: Keine Unterbezirke gefunden oder keine Daten
vorhanden.")

```

Listing 71: Skript Testfall 5 Prompt 2 Ausführung 1

```

1 import pandas as pd
2
3 # Pfad zur Excel-Datei
4 excel_file = 'Fallzahlen&HZ 2014-2023.xlsx'
5
6 # Name des Sheets
7 sheet_name = 'Fallzahlen_2023'
8
9 # Liste der Oberbezirke
10 oberbezirke = [
11     'Mitte',
12     'Friedrichshain-Kreuzberg',
13     'Pankow',
14     'Charlottenburg-Wilmersdorf',
15     'Spandau',
16     'Steglitz-Zehlendorf',
17     'Tempelhof-Schöneberg',
18     'Neukölln',
19     'Treptow-Köpenick',
20     'Marzahn-Hellersdorf',
21     'Lichtenberg',
22     'Reinickendorf'
23 ]
24
25 # Lesen des Excel-Sheets
26 df = pd.read_excel(excel_file, sheet_name=sheet_name)
27
28 # Entfernen der letzten zwei Zeilen (Gesamtwerte)
29 df = df.iloc[:-2].reset_index(drop=True)
30
31 # Initialisierung
32 current_oberbezirk = None

```

```

33 unterbezirke_dict = {ober: [] for ober in oberbezirke}
34
35 # Iteration über die Zeilen des DataFrames
36 for index, row in df.iterrows():
37     bezirk_name = row['Bezeichnung (Bezirksregion)']
38
39     if bezirk_name in oberbezirke:
40         # Aktueller Oberbezirk festlegen
41         current_oberbezirk = bezirk_name
42     else:
43         if current_oberbezirk:
44             # Unterbezirk zuweisen
45             unterbezirke_dict[current_oberbezirk].append({
46                 'Bezeichnung': bezirk_name,
47                 'Raub': row['Raub']
48             })
49         else:
50             # Falls kein Oberbezirk festgelegt ist, überspringen
51             continue
52
53 # Ermittlung des Unterbezirks mit den meisten Raubdelikten für jeden
54 # Oberbezirk
55 top_unterbezirke = {}
56
57 for oberbezirk, unterbezirke in unterbezirke_dict.items():
58     if unterbezirke:
59         # Finden des Unterbezirks mit dem maximalen 'Raub' Wert
60         top_unterbezirk = max(unterbezirke, key=lambda x: x['Raub'])
61         top_unterbezirke[oberbezirk] = {
62             'Unterbezirk': top_unterbezirk['Bezeichnung'],
63             'Raubdelikte': top_unterbezirk['Raub']
64         }
65     else:
66         top_unterbezirke[oberbezirk] = {
67             'Unterbezirk': None,
68             'Raubdelikte': 0
69         }
70
71 # Ausgabe der Ergebnisse
72 print("Unterbezirk mit den meisten Raubdelikten pro Oberbezirk (2023):\n")
73 for oberbezirk, daten in top_unterbezirke.items():
74     unterbezirk = daten['Unterbezirk']
75     raub = daten['Raubdelikte']
76     if unterbezirk:
77         print(f"{oberbezirk}: {unterbezirk} ({raub} Raubdelikte)")
78     else:
79         print(f"{oberbezirk}: Kein Unterbezirk gefunden.")

```

Listing 72: Skript Testfall 5 Prompt 2 Ausführung 2

```

1 import pandas as pd
2
3 # Pfad zur Excel-Datei

```

```

4 excel_datei = 'Fallzahlen&HZ 2014-2023.xlsx'
5
6 # Name des Sheets
7 sheet_name = 'Fallzahlen_2023'
8
9 # Liste der Oberbezirke
10 oberbezirke = [
11     'Mitte',
12     'Friedrichshain-Kreuzberg',
13     'Pankow',
14     'Charlottenburg-Wilmersdorf',
15     'Spandau',
16     'Steglitz-Zehlendorf',
17     'Tempelhof-Schöneberg',
18     'Neukölln',
19     'Treptow-Köpenick',
20     'Marzahn-Hellersdorf',
21     'Lichtenberg',
22     'Reinickendorf'
23 ]
24
25 try:
26     # Excel-Datei einlesen
27     df = pd.read_excel(excel_datei, sheet_name=sheet_name)
28
29     # Entfernen der letzten zwei Zeilen (Gesamtwerte)
30     df = df.iloc[:-2]
31
32     # Sicherstellen, dass die relevanten Spalten vorhanden sind
33     erforderliche_spalten = ['Bezeichnung (Bezirksregion)', 'Raub']
34     for spalte in erforderliche_spalten:
35         if spalte not in df.columns:
36             raise ValueError(f"Die erforderliche Spalte '{spalte}' wurde
37 nicht gefunden.")
38
39     # Erstellen einer neuen Spalte 'Oberbezirk', die den aktuellen Oberbezirk
40     # enthält
41     df['Oberbezirk'] = df['Bezeichnung (Bezirksregion)'].where(df['
42 Bezeichnung (Bezirksregion)'].isin(oberbezirke))
43     df['Oberbezirk'] = df['Oberbezirk'].ffill()
44
45     # Filtern der Unterbezirke (Ausschluss der Oberbezirke)
46     unterbezirke = df[~df['Bezeichnung (Bezirksregion)'].isin(oberbezirke)].
47     copy()
48
49     # Um sicherzustellen, dass die 'Raub' Spalte numerisch ist
50     unterbezirke['Raub'] = pd.to_numeric(unterbezirke['Raub'], errors='coerce
51 ').fillna(0).astype(int)
52
53     # Gruppieren nach 'Oberbezirk' und Finden des Unterbezirks mit den
54     # meisten Raubdelikten
55     idx_max_raub = unterbezirke.groupby('Oberbezirk')['Raub'].idxmax()

```

```

50     max_raub_unterbezirke = unterbezirke.loc[idx_max_raub]
51
52     # Ausgabe der Ergebnisse
53     print("Unterbezirk mit den meisten Raubdelikten pro Oberbezirk (2023):\n"
54         )
55     for _, row in max_raub_unterbezirke.iterrows():
56         print(f"Oberbezirk: {row['Oberbezirk']}")
57         print(f"    Unterbezirk: {row['Bezeichnung (Bezirksregion)']}")
58         print(f"    Anzahl Raubdelikte: {row['Raub']}\n")
59
60 except FileNotFoundError:
61     print(f"Die Datei '{excel_datei}' wurde nicht gefunden. Bitte überprüfe
62         den Pfad.")
63 except ValueError as ve:
64     print(f"Fehler: {ve}")
65 except Exception as e:
66     print(f"Ein unerwarteter Fehler ist aufgetreten: {e}")

```

Listing 73: Skript Testfall 5 Prompt 2 Ausführung 3

```

1  import pandas as pd
2
3  def find_top_raub_unterbezirke(excel_file, sheet_name):
4      """
5      Liest die Daten aus dem angegebenen Excel-Sheet und ermittelt für jeden
6      Oberbezirk
7
8      :param excel_file: Pfad zur Excel-Datei
9      :param sheet_name: Name des Sheets, das ausgelesen werden soll
10     :return: Dictionary mit Oberbezirken als Schlüsseln und den
11             entsprechenden
12             Unterbezirken mit den meisten Raubdelikten als Werten
13     """
14     # Liste der Oberbezirke
15     oberbezirke = [
16         'Mitte',
17         'Friedrichshain-Kreuzberg',
18         'Pankow',
19         'Charlottenburg-Wilmersdorf',
20         'Spandau',
21         'Steglitz-Zehlendorf',
22         'Tempelhof-Schöneberg',
23         'Neukölln',
24         'Treptow-Köpenick',
25         'Marzahn-Hellersdorf',
26         'Lichtenberg',
27         'Reinickendorf'
28     ]
29
30     try:
31         # Excel-Datei einlesen
32         df = pd.read_excel(excel_file, sheet_name=sheet_name, engine='

```



```

openpyxl')
32 except FileNotFoundError:
33     print(f"Fehler: Die Datei '{excel_file}' wurde nicht gefunden.")
34     return
35 except Exception as e:
36     print(f"Ein Fehler ist aufgetreten: {e}")
37     return
38
39 # Die letzten zwei Zeilen entfernen (Gesamtwerte)
40 df = df.iloc[:-2].reset_index(drop=True)
41
42 # Initialisiere ein Dictionary, um die Unterbezirke für jeden Oberbezirk
43 # zu speichern
44 oberbezirk_dict = {ober: [] for ober in oberbezirke}
45
46 current_oberbezirk = None
47
48 # Durchlaufe jede Zeile des DataFrames
49 for idx, row in df.iterrows():
50     bezirk_name = row['Bezeichnung (Bezirksregion)']
51
52     if bezirk_name in oberbezirke:
53         # Aktuellen Oberbezirk setzen
54         current_oberbezirk = bezirk_name
55         continue # Weiter zur nächsten Zeile
56
57     # Wenn kein aktueller Oberbezirk gesetzt ist, überspringe die Zeile
58     if current_oberbezirk is None:
59         continue
60
61     # Extrahiere den Raub-Wert
62     raub_count = row.get('Raub')
63
64     # Überprüfen, ob der Raub-Wert gültig ist
65     if pd.isna(raub_count):
66         raub_count = 0 # Oder eine andere Standardbehandlung
67
68     # Füge den Unterbezirk und den Raub-Wert dem aktuellen Oberbezirk
69     # hinzu
70     unterbezirk = {
71         'Name': bezirk_name,
72         'Raub': raub_count
73     }
74     oberbezirk_dict[current_oberbezirk].append(unterbezirk)
75
76 # Ergebnis-Dictionary, das die Top-Unterbezirke speichert
77 top_unterbezirke = {}
78
79 for ober, unterbezirke in oberbezirk_dict.items():
80     if not unterbezirke:
81         top_unterbezirke[ober] = None
82         continue

```

```

81
82     # Finde den Unterbezirk mit den meisten Raubdelikten
83     top_unter = max(unterbezirke, key=lambda x: x['Raub'])
84     top_unterbezirke[ober] = top_unter
85
86     return top_unterbezirke
87
88 def main():
89     # Pfad zur Excel-Datei
90     excel_file = 'Fallzahlen&HZ 2014-2023.xlsx'
91
92     # Name des Sheets
93     sheet_name = 'Fallzahlen_2023'
94
95     # Finde die Top-Unterbezirke
96     top_unterbezirke = find_top_raub_unterbezirke(excel_file, sheet_name)
97
98     if not top_unterbezirke:
99         print("Keine Daten gefunden oder ein Fehler ist aufgetreten.")
100        return
101
102    # Ausgabe der Ergebnisse
103    print("Unterbezirke mit den meisten Raubdelikten pro Oberbezirk:\n")
104    for ober, unter in top_unterbezirke.items():
105        if unter:
106            print(f"Oberbezirk: {ober}")
107            print(f"    Unterbezirk: {unter['Name']}")
108            print(f"    Raubdelikte: {unter['Raub']}\n")
109        else:
110            print(f"Oberbezirk: {ober} hat keine Unterbezirke oder keine
111Raubdelikte.\n")
112
113 if __name__ == "__main__":
114     main()

```

Listing 74: Skript Testfall 5 Prompt 2 Ausführung 4

```

1 import pandas as pd
2
3 # Definiere den Dateipfad und den Sheet-Namen
4 excel_datei = 'Fallzahlen&HZ 2014-2023.xlsx'
5 sheet_name = 'Fallzahlen_2023'
6
7 # Liste der Oberbezirke
8 oberbezirke = [
9     'Mitte',
10    'Friedrichshain-Kreuzberg',
11    'Pankow',
12    'Charlottenburg-Wilmersdorf',
13    'Spandau',
14    'Steglitz-Zehlendorf',
15    'Tempelhof-Schöneberg',
16    'Neukölln',

```

```

17     'Treptow-Köpenick',
18     'Marzahn-Hellersdorf',
19     'Lichtenberg',
20     'Reinickendorf'
21 ]
22
23 # Lese das Excel-Sheet ein
24 df = pd.read_excel(excel_datei, sheet_name=sheet_name)
25
26 # Entferne die letzten zwei Zeilen (Gesamtwerte)
27 df = df.iloc[:-2].reset_index(drop=True)
28
29 # Bereinige die Spaltennamen (entferne Zeilenumbrüche und Leerzeichen)
30 df.columns = df.columns.str.replace('\n', ' ').str.strip()
31
32 # Stelle sicher, dass die 'Raub' Spalte numerisch ist
33 df['Raub'] = pd.to_numeric(df['Raub'], errors='coerce')
34
35 # Initialisiere ein Dictionary, um die Ergebnisse zu speichern
36 ergebnisse = {}
37
38 # Variable, um den aktuellen Oberbezirk zu verfolgen
39 aktueller_oberbezirk = None
40
41 # Iteriere über die DataFrame-Zeilen
42 for index, row in df.iterrows():
43     bezirk_bezeichnung = row['Bezeichnung (Bezirksregion)']
44
45     if bezirk_bezeichnung in oberbezirke:
46         # Aktuellen Oberbezirk setzen
47         aktueller_oberbezirk = bezirk_bezeichnung
48         # Initialisiere eine Liste für die Unterbezirke des aktuellen
49         # Oberbezirks
50         ergebnisse[aktueller_oberbezirk] = []
51     else:
52         if aktueller_oberbezirk:
53             # Füge den Unterbezirk zu der Liste des aktuellen Oberbezirks
54             # hinzu
55             ergebnisse[aktueller_oberbezirk].append({
56                 'Unterbezirk': bezirk_bezeichnung,
57                 'Raub': row['Raub']
58             })
59
60 # Jetzt finde für jeden Oberbezirk den Unterbezirk mit den meisten
61 # Raubdelikten
62 for oberbezirk, unterbezirk_list in ergebnisse.items():
63     if not unterbezirk_list:
64         print(f"Oberbezirk '{oberbezirk}' hat keine Unterbezirke.")
65         continue
66
67     # Finde den Unterbezirk mit dem maximalen Raub-Wert
68     max_raub_unterbezirk = max(unterbezirk_list, key=lambda x: x['Raub'])

```

```

66
67     print(f"Oberbezirk: {oberbezirk}")
68     print(f"    Unterbezirk mit den meisten Raubdelikten: {max_raub_unterbezirk
69         ['Unterbezirk']} ({max_raub_unterbezirk['Raub']} Raubdelikte)")
    print()

```

Listing 75: Skript Testfall 5 Prompt 2 Ausführung 5

```

1  import pandas as pd
2
3  # Schritt 1: Excel-Datei und Sheet definieren
4  file_path = 'Fallzahlen&HZ 2014-2023.xlsx'
5  sheet_name = 'Fallzahlen_2023'
6
7  # Schritt 2: Daten aus dem Excel-Sheet in ein Pandas DataFrame laden
8  try:
9      df = pd.read_excel(file_path, sheet_name=sheet_name)
10 except FileNotFoundError:
11     print(f"Die Datei {file_path} wurde nicht gefunden.")
12     exit(1)
13 except ValueError:
14     print(f"Das Sheet '{sheet_name}' existiert nicht in der Datei {file_path}
15         .")
16     exit(1)
17
18 # Schritt 3: Notwendige Spalten auswählen
19 try:
20     df = df[['Bezeichnung (Bezirksregion)', 'Raub']]
21 except KeyError as e:
22     print(f"Die erforderliche Spalte wurde nicht gefunden: {e}")
23     exit(1)
24
25 # Schritt 4: Die letzten zwei Zeilen entfernen (Gesamtwerte)
26 df = df.iloc[:-2]
27
28 # Schritt 5: Liste der Oberbezirke definieren
29 oberbezirke = [
30     'Mitte',
31     'Friedrichshain-Kreuzberg',
32     'Pankow',
33     'Charlottenburg-Wilmersdorf',
34     'Spandau',
35     'Steglitz-Zehlendorf',
36     'Tempelhof-Schöneberg',
37     'Neukölln',
38     'Treptow-Köpenick',
39     'Marzahn-Hellersdorf',
40     'Lichtenberg',
41     'Reinickendorf'
42 ]
43
44 # Schritt 6: Oberbezirke identifizieren und Unterbezirke zuweisen
45 # Eine neue Spalte 'Oberbezirk' erstellen, die nur dann gesetzt ist, wenn die

```

```

    Zeile ein Oberbezirk ist
45 df['Oberbezirk'] = df['Bezeichnung (Bezirksregion)'].where(df['Bezeichnung (
    Bezirksregion)'].isin(oberbezirke))
46
47 # Die 'Oberbezirk'-Spalte nach unten füllen, sodass jeder Unterbezirk seinen
    zugehörigen Oberbezirk hat
48 df['Oberbezirk'] = df['Oberbezirk'].ffill()
49
50 # Schritt 7: Unterbezirke aus den Daten filtern (Zeilen, die keine
    Oberbezirke sind)
51 df_unterbezirke = df[~df['Bezeichnung (Bezirksregion)'].isin(oberbezirke)].
    copy()
52
53 # Schritt 8: Für jeden Oberbezirk den Unterbezirk mit dem höchsten 'Raub'
    ermitteln
54 # Gruppieren nach 'Oberbezirk' und Finden des Index mit maximalem 'Raub' in
    jeder Gruppe
55 idx = df_unterbezirke.groupby('Oberbezirk')['Raub'].idxmax()
56
57 # Das finale DataFrame erstellen mit den gewünschten Spalten
58 result_df = df_unterbezirke.loc[idx, ['Oberbezirk', 'Bezeichnung (
    Bezirksregion)', 'Raub']].copy()
59
60 # Spalten umbenennen für bessere Lesbarkeit
61 result_df.rename(columns={'Bezeichnung (Bezirksregion)': 'Unterbezirk'},
    inplace=True)
62
63 # Schritt 9: Das finale DataFrame ausgeben
64 print(result_df.reset_index(drop=True))

```

Listing 76: Skript Testfall 5 Prompt 3 Ausführung 1

```

1 import pandas as pd
2
3 # 1. Die Daten des Sheets 'Fallzahlen_2023' auslesen
4 file_path = 'Fallzahlen&HZ 2014-2023.xlsx'
5 sheet_name = 'Fallzahlen_2023'
6
7 # 2. Die Daten in einem Pandas Dataframe speichern
8 df = pd.read_excel(file_path, sheet_name=sheet_name)
9
10 # 3. Das Dataframe auf die notwendigen Spalten reduzieren: 'Bezeichnung (
    Bezirksregion)' und 'Raub'
11 df = df[['Bezeichnung (Bezirksregion)', 'Raub']]
12
13 # 4. Unnötige Zeilen entfernen: Die letzten zwei Zeilen der Tabelle sind
    Gesamtwerte und sollen entfernt werden.
14 df = df.iloc[:-2]
15
16 # 5. Die Oberbezirke in der Spalte 'Bezeichnung (Bezirksregion)'
    identifizieren
17 oberbezirke = [
18     'Mitte',

```

```

19     'Friedrichshain-Kreuzberg',
20     'Pankow',
21     'Charlottenburg-Wilmersdorf',
22     'Spandau',
23     'Steglitz-Zehlendorf',
24     'Tempelhof-Schöneberg',
25     'Neukölln',
26     'Treptow-Köpenick',
27     'Marzahn-Hellersdorf',
28     'Lichtenberg',
29     'Reinickendorf'
30 ]
31
32 # Erstellen einer neuen Spalte 'Oberbezirk', die den aktuellen Oberbezirk
    enthält
33 df['Oberbezirk'] = df['Bezeichnung (Bezirksregion)'].where(df['Bezeichnung (
    Bezirksregion)'].isin(oberbezirke))
34 df['Oberbezirk'] = df['Oberbezirk'].ffill()
35
36 # 6. Die Unterbezirke identifizieren, indem Oberbezirke selbst ausgeschlossen
    werden
37 df_unter = df[~df['Bezeichnung (Bezirksregion)'].isin(oberbezirke)].copy()
38
39 # Sicherstellen, dass die 'Raub'-Spalte numerisch ist
40 df_unter['Raub'] = pd.to_numeric(df_unter['Raub'], errors='coerce')
41
42 # Entfernen von Zeilen mit fehlenden 'Raub'-Werten
43 df_unter = df_unter.dropna(subset=['Raub'])
44
45 # Für jeden Oberbezirk den Unterbezirk mit dem höchsten Wert in der Spalte '
    Raub' identifizieren
46 idx_max_raub = df_unter.groupby('Oberbezirk')['Raub'].idxmax()
47 df_max_raub = df_unter.loc[idx_max_raub]
48
49 # 7. Die Ergebnisse in einem neuen Dataframe speichern mit den Spalten: '
    Oberbezirk', 'Unterbezirk', 'Raub'
50 final_df = df_max_raub[['Oberbezirk', 'Bezeichnung (Bezirksregion)', 'Raub'
    ]].rename(columns={
51     'Bezeichnung (Bezirksregion)': 'Unterbezirk'
52 }).reset_index(drop=True)
53
54 # 8. Das finale Dataframe ausgeben
55 print(final_df)

```

Listing 77: Skript Testfall 5 Prompt 3 Ausführung 2

```

1 import pandas as pd
2
3 # Schritt 1: Die Daten des Sheets 'Fallzahlen_2023' auslesen
4 excel_file = 'Fallzahlen&HZ 2014-2023.xlsx'
5 sheet_name = 'Fallzahlen_2023'
6
7 # Schritt 2: Die Daten in einem Pandas DataFrame speichern

```

```

8 df = pd.read_excel(excel_file, sheet_name=sheet_name)
9
10 # Schritt 3: Das DataFrame auf die notwendigen Spalten reduzieren
11 necessary_columns = ['Bezeichnung (Bezirksregion)', 'Raub']
12 df = df[necessary_columns]
13
14 # Schritt 4: Unnötige Zeilen entfernen (letzte zwei Zeilen)
15 df = df.iloc[:-2].reset_index(drop=True)
16
17 # Schritt 5: Die Oberbezirke definieren
18 oberbezirke = [
19     'Mitte', 'Friedrichshain-Kreuzberg', 'Pankow',
20     'Charlottenburg-Wilmersdorf', 'Spandau', 'Steglitz-Zehlendorf',
21     'Tempelhof-Schöneberg', 'Neukölln', 'Treptow-Köpenick',
22     'Marzahn-Hellersdorf', 'Lichtenberg', 'Reinickendorf'
23 ]
24
25 # Hinzufügen einer neuen Spalte zur Identifizierung des aktuellen Oberbezirks
26 df['Oberbezirk'] = None
27 current_oberbezirk = None
28
29 for index, row in df.iterrows():
30     bezirk = row['Bezeichnung (Bezirksregion)']
31     if bezirk in oberbezirke:
32         current_oberbezirk = bezirk
33         df.at[index, 'Oberbezirk'] = current_oberbezirk
34     else:
35         df.at[index, 'Oberbezirk'] = current_oberbezirk
36
37 # Schritt 6: Unterbezirke identifizieren (Zeilen, die keine Oberbezirke sind)
38 unterbezirke_df = df[~df['Bezeichnung (Bezirksregion)'].isin(oberbezirke)].
39     copy()
40
41 # Sicherstellen, dass 'Raub' als numerischer Wert vorliegt
42 unterbezirke_df['Raub'] = pd.to_numeric(unterbezirke_df['Raub'], errors='
43     coerce')
44
45 # Schritt 6: Für jeden Oberbezirk den Unterbezirk mit dem höchsten 'Raub'
46     finden
47 max_raub_df = unterbezirke_df.loc[unterbezirke_df.groupby('Oberbezirk')['Raub
48     '].idxmax()]
49
50 # Schritt 7: Die Ergebnisse in einem neuen DataFrame speichern
51 final_df = max_raub_df[['Oberbezirk', 'Bezeichnung (Bezirksregion)', 'Raub'
52     ']].rename(
53     columns={
54         'Bezeichnung (Bezirksregion)': 'Unterbezirk',
55         'Raub': 'Raub'
56     }
57 ).reset_index(drop=True)
58
59 # Schritt 8: Das finale DataFrame ausgeben

```

```
55 print(final_df)
```

Listing 78: Skript Testfall 5 Prompt 3 Ausführung 3

```
1 import pandas as pd
2
3 # Schritt 1: Excel-Datei lesen und das Sheet 'Fallzahlen_2023' laden
4 datei_name = 'Fallzahlen&HZ 2014-2023.xlsx'
5 sheet_name = 'Fallzahlen_2023'
6
7 # Lese die Excel-Datei
8 try:
9     df = pd.read_excel(datei_name, sheet_name=sheet_name)
10    print(f"Sheet '{sheet_name}' erfolgreich geladen.")
11 except FileNotFoundError:
12    print(f"Die Datei '{datei_name}' wurde nicht gefunden.")
13    exit(1)
14 except ValueError:
15    print(f"Das Sheet '{sheet_name}' existiert nicht in der Datei '{datei_name}'.")
16    exit(1)
17
18 # Schritt 2: Daten in einem Pandas DataFrame speichern
19 # (Bereits durch pd.read_excel erledigt)
20
21 # Schritt 3: DataFrame auf notwendige Spalten reduzieren
22 notwendige_spalten = ['Bezeichnung (Bezirksregion)', 'Raub']
23 # Überprüfe, ob alle notwendigen Spalten vorhanden sind
24 fehlende_spalten = [spalte for spalte in notwendige_spalten if spalte not in df.columns]
25 if fehlende_spalten:
26    print(f"Die folgenden benötigten Spalten fehlen im DataFrame: {fehlende_spalten}")
27    exit(1)
28
29 df = df[notwendige_spalten]
30 print("DataFrame auf notwendige Spalten reduziert.")
31
32 # Schritt 4: Letzte zwei Zeilen entfernen (Gesamtwerte)
33 df = df.iloc[:-2].reset_index(drop=True)
34 print("Letzte zwei Zeilen (Gesamtwerte) entfernt.")
35
36 # Schritt 5: Oberbezirke identifizieren
37 oberbezirke = [
38     'Mitte',
39     'Friedrichshain-Kreuzberg',
40     'Pankow',
41     'Charlottenburg-Wilmersdorf',
42     'Spandau',
43     'Steglitz-Zehlendorf',
44     'Tempelhof-Schöneberg',
45     'Neukölln',
46     'Treptow-Köpenick',
```



```

47     'Marzahn-Hellersdorf',
48     'Lichtenberg',
49     'Reinickendorf'
50 ]
51
52 # Füge eine neue Spalte 'Oberbezirk' hinzu, um jedem Unterbezirk den zugehö-
    rigen Oberbezirk zuzuweisen
53 df['Oberbezirk'] = None
54 current_oberbezirk = None
55
56 for idx, row in df.iterrows():
57     bezirk = row['Bezeichnung (Bezirksregion)']
58     if bezirk in oberbezirke:
59         current_oberbezirk = bezirk
60         df.at[idx, 'Oberbezirk'] = current_oberbezirk
61     else:
62         df.at[idx, 'Oberbezirk'] = current_oberbezirk
63
64 print("Oberbezirke zugewiesen.")
65
66 # Schritt 6: Oberbezirke von Unterbezirken trennen
67 # Annahme: Oberbezirke selbst sind keine Unterbezirke und sollten nicht berü-
    cksichtigt werden
68 df_unterbezirke = df[~df['Bezeichnung (Bezirksregion)'].isin(oberbezirke)].
    copy()
69 print("Unterbezirke von Oberbezirken getrennt.")
70
71 # Überprüfe, ob jedem Unterbezirk ein Oberbezirk zugewiesen wurde
72 if df_unterbezirke['Oberbezirk'].isnull().any():
73     print("Warnung: Einige Unterbezirke haben keinen zugewiesenen Oberbezirk.
    ")
74
75 # Schritt 7: Für jeden Oberbezirk den Unterbezirk mit dem höchsten 'Raub'
    identifizieren
76 # Zuerst stelle sicher, dass die 'Raub'-Spalte numerisch ist
77 df_unterbezirke['Raub'] = pd.to_numeric(df_unterbezirke['Raub'], errors='
    coerce')
78
79 # Entferne Zeilen mit fehlenden 'Raub'-Werten
80 df_unterbezirke = df_unterbezirke.dropna(subset=['Raub'])
81
82 # Gruppiere nach 'Oberbezirk' und finde den Unterbezirk mit dem maximalen '
    Raub'
83 max_raub_unterbezirke = df_unterbezirke.loc[df_unterbezirke.groupby('
    Oberbezirk')['Raub'].idxmax()].reset_index(drop=True)
84
85 # Schritt 8: Ergebnis in einem neuen DataFrame speichern und formatieren
86 final_df = max_raub_unterbezirke[['Oberbezirk', 'Bezeichnung (Bezirksregion)',
    'Raub']].rename(
87     columns={
88         'Bezeichnung (Bezirksregion)': 'Unterbezirk',
89         'Raub': 'Raub'

```

```

90     }
91 )
92
93 # Optional: Sortiere das finale DataFrame nach 'Oberbezirk' für bessere Ü
    bersichtlichkeit
94 final_df = final_df.sort_values(by='Oberbezirk').reset_index(drop=True)
95
96 # Schritt 9: Finale Ergebnisse anzeigen
97 print("\nUnterbezirke mit den höchsten 'Raub'-Werten je Oberbezirk:")
98 print(final_df)
99
100 # Optional: Speichere das Ergebnis in eine neue Excel-Datei
101 output_datei = 'Max_Raub_Unterbezirke_2023.xlsx'
102 final_df.to_excel(output_datei, index=False)
103 print(f"\nErgebnisse wurden in '{output_datei}' gespeichert.")

```

Listing 79: Skript Testfall 5 Prompt 3 Ausführung 4

```

1 import pandas as pd
2
3 # Schritt 1: Excel-Datei und das spezifische Sheet laden
4 excel_file = 'Fallzahlen&HZ 2014-2023.xlsx'
5 sheet_name = 'Fallzahlen_2023'
6
7 # Schritt 2: Daten in einem Pandas DataFrame speichern
8 df = pd.read_excel(excel_file, sheet_name=sheet_name)
9
10 # Schritt 3: DataFrame auf die notwendigen Spalten reduzieren
11 selected_columns = ['Bezeichnung (Bezirksregion)', 'Raub']
12 df = df[selected_columns]
13
14 # Schritt 4: Die letzten zwei Zeilen entfernen (Gesamtwerte)
15 df = df.iloc[:-2].reset_index(drop=True)
16
17 # Schritt 5: Liste der Oberbezirke definieren
18 oberbezirke = [
19     'Mitte',
20     'Friedrichshain-Kreuzberg',
21     'Pankow',
22     'Charlottenburg-Wilmersdorf',
23     'Spandau',
24     'Steglitz-Zehlendorf',
25     'Tempelhof-Schöneberg',
26     'Neukölln',
27     'Treptow-Köpenick',
28     'Marzahn-Hellersdorf',
29     'Lichtenberg',
30     'Reinickendorf'
31 ]
32
33 # Initialisierung
34 results = []
35 current_oberbezirk = None

```

```

36
37 # Schritt 6: Durch das DataFrame iterieren und Unterbezirke zuordnen
38 for index, row in df.iterrows():
39     bezirk = row['Bezeichnung (Bezirksregion)']
40     raub = row['Raub']
41
42     if bezirk in oberbezirke:
43         # Aktuellen Oberbezirk setzen
44         current_oberbezirk = bezirk
45     else:
46         if current_oberbezirk:
47             # Unterbezirk und Raub-Wert hinzufügen
48             results.append({
49                 'Oberbezirk': current_oberbezirk,
50                 'Unterbezirk': bezirk,
51                 'Raub': raub
52             })
53
54 # In einen neuen DataFrame umwandeln
55 results_df = pd.DataFrame(results)
56
57 # Schritt 6 (Fortsetzung): Für jeden Oberbezirk den Unterbezirk mit dem höchsten 'Raub' finden
58 max_raub_df = results_df.loc[results_df.groupby('Oberbezirk')['Raub'].idxmax()
59                             ].reset_index(drop=True)
60
61 # Schritt 7: Spalten umbenennen (falls nötig) und finalen DataFrame erstellen
62 final_df = max_raub_df[['Oberbezirk', 'Unterbezirk', 'Raub']]
63
64 # Schritt 8: Finalen DataFrame ausgeben
65 print(final_df)

```

Listing 80: Skript Testfall 5 Prompt 3 Ausführung 5

11.3.6 Testfall 6

```

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&HZ 2014-2023.xlsx"
13 df = pd.read_excel(file, sheet_name='Fallzahlen_2023')
14
15 # Prompt erstellen: CSV-Daten einbetten + Anweisung
16 prompt = f"""

```

```

17 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 zurü
    ck gibt.
18 Hier sind die Daten des Sheets 'Fallzahlen_2023' als Beispiel: {df}
19 ""
20
21 # Anfrage an die OpenAI Chat-API
22 response = client.chat.completions.create(model="o1-mini",
23 messages=[
24 {"role": "user", "content": prompt}
25 ],)
26
27 response_file = path+"testcases/testcase6/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 81: OpenAI API Request Skript

```

1 import pandas as pd
2
3 def get_top_unterbezirke(excel_file, sheets_prefix='Fallzahlen_', start_year
    =2014, end_year=2023, top_n=10):
4     # Liste zur Speicherung der DataFrames aus allen Jahren
5     df_list = []
6
7     for year in range(start_year, end_year + 1):
8         sheet_name = f"{sheets_prefix}{year}"
9         try:
10             # Lade das jeweilige Sheet
11             df = pd.read_excel(excel_file, sheet_name=sheet_name)
12
13             # Überprüfe, ob die notwendigen Spalten vorhanden sind
14             if 'LOR-Schlüssel (Bezirksregion)' in df.columns and 'Straftaten
                insgesamt' in df.columns:
15                 # Wähle relevante Spalten
16                 df_subset = df[['LOR-Schlüssel (Bezirksregion)', 'Bezeichnung
                    (Bezirksregion)', 'Straftaten insgesamt']].copy()
17
18                 # Falls es weitere Bereinigungen benötigt, kann dies hier
                eingefügt werden
19
20                 df_list.append(df_subset)
21             else:
22                 print(f"Wichtige Spalten fehlen im Sheet {sheet_name}.")
23         except Exception as e:
24             print(f"Fehler beim Laden des Sheets {sheet_name}: {e}")
25

```

```

26     if not df_list:
27         print("Keine Daten zum Verarbeiten gefunden.")
28         return
29
30     # Kombiniere alle DataFrames
31     combined_df = pd.concat(df_list, ignore_index=True)
32
33     # Gruppieren nach Unterbezirk und summiere die Straftaten insgesamt
34     aggregated_df = combined_df.groupby(['LOR-Schlüssel (Bezirksregion)', '
35     Bezeichnung (Bezirksregion)'])['Straftaten insgesamt'].sum().reset_index()
36
37     # Sortiere nach den meisten Straftaten und wähle die Top N
38     top_unterbezirke = aggregated_df.sort_values(by='Straftaten insgesamt',
39     ascending=False).head(top_n)
40
41     return top_unterbezirke
42
43 if __name__ == "__main__":
44     excel_file = 'Fallzahlen&HZ 2014-2023.xlsx' # Pfad zu deiner Excel-Datei
45     top_n = 10 # Anzahl der Top-Unterbezirke
46
47     top_unterbezirke = get_top_unterbezirke(excel_file, top_n=top_n)
48
49     if top_unterbezirke is not None:
50         print(f"Top {top_n} Unterbezirke mit den meisten Straftaten insgesamt
51         (2014-2023):")
52         print(top_unterbezirke.to_string(index=False))

```

Listing 82: Skript Testfall 6 Prompt 1 Ausführung 1

```

1 import pandas as pd
2
3 def get_top_10_unterbezirke(excel_file):
4     # Liste der Jahre von 2014 bis 2023
5     jahre = list(range(2014, 2024))
6
7     # Initialisiere ein leeres DataFrame für die aggregierten Daten
8     aggregierte_daten = pd.DataFrame()
9
10    # Iteriere über jede Jahres-Sheet
11    for jahr in jahre:
12        sheet_name = f'Fallzahlen_{jahr}'
13        try:
14            # Lese das aktuelle Sheet
15            df = pd.read_excel(excel_file, sheet_name=sheet_name)
16
17            # Überprüfe, ob die notwendigen Spalten vorhanden sind
18            if 'Bezeichnung (Bezirksregion)' not in df.columns or 'Straftaten
19            insgesamt' not in df.columns:
20                print(f"Wichtige Spalten fehlen im Sheet {sheet_name}. Ü
21                berspringe dieses Sheet.")
22                continue

```

```

22         # Auswahl der relevanten Spalten
23         # Annahme: 'Straftaten insgesamt' ist die Spalte mit den gesamten
        Straftaten
24         # Passen Sie die Spaltennamen an, falls sie sich unterscheiden
25         df_relevant = df[['Bezeichnung (Bezirksregion)', 'Straftaten
        insgesamt']]
26
27         # Gruppieren nach Unterbezirk und summiere die Straftaten
28         df_grouped = df_relevant.groupby('Bezeichnung (Bezirksregion)',
        as_index=False).sum()
29
30         # Füge die Daten dem aggregierten DataFrame hinzu
31         if aggregierte_daten.empty:
32             aggregierte_daten = df_grouped
33         else:
34             aggregierte_daten = aggregierte_daten.merge(df_grouped, on='
        Bezeichnung (Bezirksregion)', how='outer')
35             aggregierte_daten['Straftaten insgesamt'] = aggregierte_daten
        ['Straftaten insgesamt_x'].fillna(0) + aggregierte_daten['Straftaten
        insgesamt_y'].fillna(0)
36             aggregierte_daten.drop(['Straftaten insgesamt_x', 'Straftaten
        insgesamt_y'], axis=1, inplace=True)
37
38         except ValueError:
39             print(f"Sheet {sheet_name} nicht gefunden in der Datei. Ü
        berspringe dieses Sheet.")
40             continue
41
42         # Nach allen Sheets aggregiert, sortiere die Unterbezirke nach Straftaten
        insgesamt
43         aggregierte_daten_sorted = aggregierte_daten.sort_values(by='Straftaten
        insgesamt', ascending=False)
44
45         # Wähle die Top 10 Unterbezirke aus
46         top_10 = aggregierte_daten_sorted.head(10)
47
48         return top_10
49
50 if __name__ == "__main__":
51     excel_datei = 'Fallzahlen&HZ 2014-2023.xlsx'
52     top_10_unterbezirke = get_top_10_unterbezirke(excel_datei)
53     print("Die Top 10 Unterbezirke mit den meisten Straftaten insgesamt
        (2014-2023):\n")
54     print(top_10_unterbezirke.to_string(index=False))

```

Listing 83: Skript Testfall 6 Prompt 1 Ausführung 2

```

1 import pandas as pd
2
3 # Pfad zur Excel-Datei
4 excel_file = 'Fallzahlen&HZ 2014-2023.xlsx'
5
6 # Liste der relevanten Sheet-Namen

```

```

7 sheet_names = [f'Fallzahlen_{jahr}' for jahr in range(2014, 2024)]
8
9 # Initialisiere ein leeres DataFrame zur Speicherung aller Daten
10 all_data = pd.DataFrame()
11
12 # Iteriere über alle relevanten Sheets und füge die Daten hinzu
13 for sheet in sheet_names:
14     try:
15         # Lese das aktuelle Sheet
16         df = pd.read_excel(excel_file, sheet_name=sheet)
17
18         # Überprüfe, ob die notwendigen Spalten vorhanden sind
19         if 'Bezeichnung (Bezirksregion)' in df.columns and 'Straftaten \n-
insgesamt-' in df.columns:
20             # Wähle nur die relevanten Spalten
21             df_subset = df[['Bezeichnung (Bezirksregion)', 'Straftaten \n-
insgesamt-']]
22
23             # Entferne mögliche NaN-Werte und stelle sicher, dass 'Straftaten
\n-insgesamt-' numerisch ist
24             df_subset = df_subset.dropna(subset=['Straftaten \n-insgesamt-'])
25             df_subset['Straftaten \n-insgesamt-'] = pd.to_numeric(df_subset['
Straftaten \n-insgesamt-'], errors='coerce').fillna(0)
26
27             # Füge die Daten zum all_data DataFrame hinzu
28             all_data = pd.concat([all_data, df_subset], ignore_index=True)
29         else:
30             print(f"Warnung: Die notwendigen Spalten sind im Sheet '{sheet}'
nicht vorhanden.")
31         except Exception as e:
32             print(f"Fehler beim Verarbeiten des Sheets '{sheet}': {e}")
33
34 # Gruppieren die Daten nach Unterbezirk und summiere die Straftaten
35 total_straftaten = all_data.groupby('Bezeichnung (Bezirksregion)')['
Straftaten \n-insgesamt-'].sum()
36
37 # Sortiere die Unterbezirke nach den meisten Straftaten
38 top_10_unterbezirke = total_straftaten.sort_values(ascending=False).head(10)
39
40 # Ausgabe der Ergebnisse
41 print("Die Top 10 Unterbezirke mit den meisten Straftaten (2014-2023):")
42 print(top_10_unterbezirke)

```

Listing 84: Skript Testfall 6 Prompt 1 Ausführung 3

```

1 import pandas as pd
2
3 # Pfad zur Excel-Datei
4 excel_file = 'Fallzahlen&HZ 2014-2023.xlsx'
5
6 # Liste der Jahre, die verarbeitet werden sollen
7 jahre = range(2014, 2024) # 2014 bis 2023
8

```

```

9 # Initialisiere ein leeres DataFrame für die Aggregation
10 gesamt_df = pd.DataFrame()
11
12 # Iteriere über jedes Jahr und lese das entsprechende Sheet
13 for jahr in jahre:
14     sheet_name = f'Fallzahlen_{jahr}'
15     try:
16         # Lese das Sheet
17         df = pd.read_excel(excel_file, sheet_name=sheet_name)
18
19         # Überprüfe, ob die notwendigen Spalten vorhanden sind
20         notwendige_spalten = ['Bezeichnung (Bezirksregion)', 'Straftaten
insgesamt']
21         if not all(spalte in df.columns for spalte in notwendige_spalten):
22             print(f"Warnung: Nicht alle notwendigen Spalten in Sheet {
sheet_name} gefunden.")
23             continue
24
25         # Wähle nur die relevanten Spalten
26         df_relevant = df[['Bezeichnung (Bezirksregion)', 'Straftaten
insgesamt']]
27
28         # Benenne die Spalten um, um das Jahr zu identifizieren
29         df_relevant = df_relevant.rename(columns={'Straftaten insgesamt': f'
Straftaten_{jahr}'})
30
31         # Setze die Bezirksregion als Index
32         df_relevant.set_index('Bezeichnung (Bezirksregion)', inplace=True)
33
34         # Füge zum Gesamt-DataFrame hinzu
35         if gesamt_df.empty:
36             gesamt_df = df_relevant
37         else:
38             gesamt_df = gesamt_df.join(df_relevant, how='outer')
39
40     except FileNotFoundError:
41         print(f"Fehler: Die Datei {excel_file} wurde nicht gefunden.")
42         exit(1)
43     except ValueError:
44         print(f"Warnung: Das Sheet {sheet_name} wurde in der Datei {
excel_file} nicht gefunden.")
45         continue
46
47 # Ersetze fehlende Werte (falls ein Bezirk in einem Jahr fehlt) mit 0
48 gesamt_df.fillna(0, inplace=True)
49
50 # Berechne die Gesamtsumme der Straftaten über alle Jahre hinweg
51 gesamt_df['Straftaten_Gesamt'] = gesamt_df.sum(axis=1)
52
53 # Sortiere die Bezirke nach der Gesamtsumme der Straftaten in absteigender
Reihenfolge
54 top10 = gesamt_df['Straftaten_Gesamt'].sort_values(ascending=False).head(10)

```



```

55
56 # Optional: Zurück in ein DataFrame für bessere Darstellung
57 top10_df = top10.reset_index()
58 top10_df.columns = ['Bezirksregion', 'Gesamt_Straftaten']
59
60 # Ausgabe der Top 10 Bezirke
61 print("Top 10 Unterbezirke mit den meisten Straftaten (2014-2023):")
62 print(top10_df)
63
64 # Optional: Speichern der Ergebnisse in eine neue Excel-Datei
65 # top10_df.to_excel('Top10_Unterbezirke_Straftaten_2014-2023.xlsx', index=
    False)

```

Listing 85: Skript Testfall 6 Prompt 1 Ausführung 4

```

1 import pandas as pd
2
3 # Pfad zur Excel-Datei
4 excel_datei = 'Fallzahlen&HZ 2014-2023.xlsx'
5
6 # Liste der relevanten Sheet-Namen
7 jahre = list(range(2014, 2024)) # 2014 bis 2023
8 sheet_namen = [f'Fallzahlen_{jahr}' for jahr in jahre]
9
10 # Initialisiere ein leeres DataFrame zur Aggregation
11 aggregate_df = pd.DataFrame()
12
13 # Iteriere über jedes Sheet und aggregiere die Daten
14 for sheet in sheet_namen:
15     try:
16         # Lese das aktuelle Sheet
17         df = pd.read_excel(excel_datei, sheet_name=sheet)
18
19         # Überprüfe, ob die notwendigen Spalten vorhanden sind
20         if 'Bezeichnung (Bezirksregion)' in df.columns and 'Straftaten -
            insgesamt-' in df.columns:
21             # Wähle die relevanten Spalten aus
22             temp_df = df[['Bezeichnung (Bezirksregion)', 'Straftaten -
                insgesamt-']].copy()
23
24             # Gruppieren nach Bezirksregion und summiere die Straftaten
25             temp_agg = temp_df.groupby('Bezeichnung (Bezirksregion)',
                as_index=False)['Straftaten -insgesamt-'].sum()
26
27             # Füge die aggregierten Daten zum Gesamt-DataFrame hinzu
28             aggregate_df = pd.concat([aggregate_df, temp_agg], ignore_index=
                True)
29         else:
30             print(f"Warnung: Die erforderlichen Spalten sind im Sheet '{sheet
                }' nicht vorhanden.")
31         except Exception as e:
32             print(f"Fehler beim Verarbeiten des Sheets '{sheet}': {e}")
33

```

```

34 # Überprüfe, ob Daten aggregiert wurden
35 if aggregate_df.empty:
36     print("Keine Daten zum Aggregieren gefunden.")
37 else:
38     # Gruppieren erneut nach Bezirksregion, um die Gesamtstraftaten über alle
39     # Jahre zu berechnen
40     gesamt_agg = aggregate_df.groupby('Bezeichnung (Bezirksregion)', as_index
41                                     =False)['Straftaten -insgesamt-'].sum()
42
43     # Sortiere die Bezirke nach den meisten Straftaten
44     top_10 = gesamt_agg.sort_values(by='Straftaten -insgesamt-', ascending=
45                                     False).head(10)
46
47     # Ausgabe der Top 10 Unterbezirke
48     print("Top 10 Unterbezirke mit den meisten Straftaten insgesamt
49     (2014-2023):")
50     print(top_10.to_string(index=False))
51
52     # Optional: Speichern der Ergebnisse in einer CSV-Datei
53     top_10.to_csv('Top_10_Unterbezirke_Straftaten_2014-2023.csv', index=False
54                  )
55     print("\nDie Ergebnisse wurden in 'Top_10_Unterbezirke_Straftaten_2014
56     -2023.csv' gespeichert.")

```

Listing 86: Skript Testfall 6 Prompt 1 Ausführung 5

```

1 import pandas as pd
2
3 # Pfad zur Excel-Datei
4 excel_datei = 'Fallzahlen&HZ 2014-2023.xlsx'
5
6 # Liste der Oberbezirke
7 oberbezirke = [
8     'Mitte',
9     'Friedrichshain-Kreuzberg',
10    'Pankow',
11    'Charlottenburg-Wilmersdorf',
12    'Spandau',
13    'Steglitz-Zehlendorf',
14    'Tempelhof-Schöneberg',
15    'Neukölln',
16    'Treptow-Köpenick',
17    'Marzahn-Hellersdorf',
18    'Lichtenberg',
19    'Reinickendorf'
20 ]
21
22 # Liste der Sheet-Namen von Fallzahlen_2014 bis Fallzahlen_2023
23 jahre = list(range(2014, 2024))
24 sheet_namen = [f'Fallzahlen_{jahr}' for jahr in jahre]
25
26 # Leere Liste zur Speicherung der DataFrames
27 df_list = []

```

```

28
29 # Iteriere über jedes Sheet und lese die Daten
30 for sheet in sheet_namen:
31     try:
32         df = pd.read_excel(excel_datei, sheet_name=sheet)
33         df_list.append(df)
34     except Exception as e:
35         print(f"Fehler beim Lesen des Sheets {sheet}: {e}")
36
37 # Zusammenführen aller DataFrames
38 gesamt_df = pd.concat(df_list, ignore_index=True)
39
40 # Entfernen der Oberbezirke und nicht zugeordneten Zeilen
41 # Bedingungen:
42 # 1. 'Bezeichnung (Bezirksregion)' nicht in oberbezirke
43 # 2. 'Bezeichnung (Bezirksregion)' enthält nicht 'nicht zuzuordnen'
44
45 # Erste Bedingung
46 bedingung_1 = ~gesamt_df['Bezeichnung (Bezirksregion)'].isin(oberbezirke)
47
48 # Zweite Bedingung: Ausschließen von Zeilen, die 'nicht zuzuordnen' enthalten
49 bedingung_2 = ~gesamt_df['Bezeichnung (Bezirksregion)'].str.contains('nicht
    zuzuordnen', case=False, na=False)
50
51 # Anwenden der Bedingungen
52 unterbezirke_df = gesamt_df[bedingung_1 & bedingung_2]
53
54 # Sicherstellen, dass die Spalte 'Straftaten -insgesamt-' numerisch ist
55 # Eventuell müssen Nicht-Nummerische Werte (z.B. NaN) behandelt werden
56 unterbezirke_df['Straftaten -insgesamt-'] = pd.to_numeric(unterbezirke_df['
    Straftaten -insgesamt-'], errors='coerce').fillna(0)
57
58 # Aggregieren der Straftaten über alle Jahre hinweg
59 aggregiert_df = unterbezirke_df.groupby('Bezeichnung (Bezirksregion)')['
    Straftaten -insgesamt-'].sum().reset_index()
60
61 # Sortieren nach absteigender Anzahl der Straftaten und Auswahl der Top 10
62 top10_df = aggregiert_df.sort_values(by='Straftaten -insgesamt-', ascending=
    False).head(10)
63
64 # Optional: Zurücksetzen des Indexes
65 top10_df = top10_df.reset_index(drop=True)
66
67 # Ausgabe des finalen DataFrames
68 print(top10_df)
69
70 # Optional: Speichern des Ergebnisses in eine neue Excel-Datei
71 # top10_df.to_excel('Top10_Unterbezirke_Straftaten.xlsx', index=False)

```

Listing 87: Skript Testfall 6 Prompt 2 Ausführung 1

```

1 import pandas as pd
2

```

```

3 # Pfad zur Excel-Datei
4 excel_file = 'Fallzahlen&HZ 2014-2023.xlsx'
5
6 # Liste der Oberbezirke
7 oberbezirke = [
8     'Mitte',
9     'Friedrichshain-Kreuzberg',
10    'Pankow',
11    'Charlottenburg-Wilmersdorf',
12    'Spandau',
13    'Steglitz-Zehlendorf',
14    'Tempelhof-Schöneberg',
15    'Neukölln',
16    'Treptow-Köpenick',
17    'Marzahn-Hellersdorf',
18    'Lichtenberg',
19    'Reinickendorf'
20 ]
21
22 # Generiere die Liste der Sheet-Namen Fallzahlen_2014 bis Fallzahlen_2023
23 jahre = range(2014, 2024)
24 sheet_names = [f'Fallzahlen_{jahr}' for jahr in jahre]
25
26 # Liste zur Speicherung der einzelnen DataFrames
27 df_list = []
28
29 # Iteriere über alle Sheet-Namen und lese die Daten
30 for sheet in sheet_names:
31     try:
32         df = pd.read_excel(excel_file, sheet_name=sheet)
33         df_list.append(df)
34         print(f'Sheet {sheet} erfolgreich gelesen.')
35     except Exception as e:
36         print(f'Fehler beim Lesen des Sheets {sheet}: {e}')
37
38 # Zusammenführen aller DataFrames in einen einzigen DataFrame
39 combined_df = pd.concat(df_list, ignore_index=True)
40 print('Alle Sheets wurden erfolgreich zusammengeführt.')
41
42 # Anzeige der Spalten zur Überprüfung (optional)
43 # print(combined_df.columns)
44
45 # Filtern der Unterbezirke:
46 # - Entfernen der Oberbezirke
47 # - Entfernen von Zeilen, die "nicht zuzuordnen" enthalten
48
49 # Bedingung für Unterbezirke:
50 # - 'Bezeichnung (Bezirksregion)' ist nicht in der Liste der Oberbezirke
51 # - 'Bezeichnung (Bezirksregion)' enthält nicht 'nicht zuzuordnen' (case
    insensitive)
52
53 filtered_df = combined_df[

```

```

54     (~combined_df['Bezeichnung (Bezirksregion)'].isin(oberbezirke)) &
55     (~combined_df['Bezeichnung (Bezirksregion)'].str.contains('nicht
        zuzuordnen', case=False, na=False))
56 ]
57
58 print(f'Anzahl der Zeilen nach Filtern: {filtered_df.shape[0]}')
59
60 # Sicherstellen, dass die Spalte 'Straftaten -insgesamt-' numerisch ist
61 # Entfernen von Tausenderpunkten und Umwandeln in Integer
62 filtered_df['Straftaten -insgesamt-'] = filtered_df['Straftaten -insgesamt-']
        .astype(str).str.replace('.', '').astype(int)
63
64 # Aggregieren der Straftaten pro Unterbezirk über alle Jahre hinweg
65 aggregated_df = filtered_df.groupby('Bezeichnung (Bezirksregion)', as_index=
        False)['Straftaten -insgesamt-'].sum()
66
67 # Sortieren nach der aggregierten Anzahl der Straftaten in absteigender
        Reihenfolge
68 sorted_df = aggregated_df.sort_values(by='Straftaten -insgesamt-', ascending=
        False)
69
70 # Auswahl der Top 10 Unterbezirke mit den meisten Straftaten
71 top_10_df = sorted_df.head(10)
72
73 # Auswahl der gewünschten Spalten
74 final_df = top_10_df[['Bezeichnung (Bezirksregion)', 'Straftaten -insgesamt-']
        ]
75
76 # Optional: Umbenennen der Spalten für bessere Verständlichkeit
77 final_df.rename(columns={
78     'Bezeichnung (Bezirksregion)': 'Unterbezirk',
79     'Straftaten -insgesamt-': 'Gesamtstraftaten'
80 }, inplace=True)
81
82 # Anzeige des finalen DataFrames
83 print('Top 10 Unterbezirke mit den meisten Straftaten:')
84 print(final_df)
85
86 # Optional: Speichern des finalen DataFrames in eine neue Excel-Datei
87 # final_df.to_excel('Top_10_Unterbezirke_Straftaten.xlsx', index=False)

```

Listing 88: Skript Testfall 6 Prompt 2 Ausführung 2

```

1 import pandas as pd
2
3 # Pfad zur Excel-Datei
4 excel_datei = 'Fallzahlen&HZ 2014-2023.xlsx'
5
6 # Liste der Oberbezirke, die ausgeschlossen werden sollen
7 oberbezirke = [
8     'Mitte', 'Friedrichshain-Kreuzberg', 'Pankow',
9     'Charlottenburg-Wilmersdorf', 'Spandau', 'Steglitz-Zehlendorf',
10    'Tempelhof-Schöneberg', 'Neukölln', 'Treptow-Köpenick',

```

```

11     'Marzahn-Hellersdorf', 'Lichtenberg', 'Reinickendorf'
12 ]
13
14 # Erstellen einer Liste der Sheet-Namen von 2014 bis 2023
15 jahreszahlen = range(2014, 2024) # 2014 bis 2023 inklusive
16 sheet_namen = [f'Fallzahlen_{jahr}' for jahr in jahreszahlen]
17
18 # Lesen aller relevanten Sheets und Speichern in einer Liste von DataFrames
19 dataframes = []
20 for sheet in sheet_namen:
21     try:
22         df = pd.read_excel(excel_datei, sheet_name=sheet, engine='openpyxl')
23         dataframes.append(df)
24         print(f"Sheet '{sheet}' erfolgreich gelesen.")
25     except Exception as e:
26         print(f"Fehler beim Lesen von Sheet '{sheet}': {e}")
27
28 # Zusammenführen aller DataFrames zu einem einzigen DataFrame
29 if dataframes:
30     kombiniertes_df = pd.concat(dataframes, ignore_index=True)
31     print("Alle Sheets erfolgreich zusammengeführt.")
32 else:
33     raise ValueError("Keine Daten zum Verarbeiten gefunden.")
34
35 # Überprüfen der Spaltennamen
36 print("Spaltennamen im kombinierten DataFrame:")
37 print(kombiniertes_df.columns.tolist())
38
39 # Bereinigung der Spaltennamen (Entfernen von Zeilenumbrüchen und Leerzeichen)
40 kombiniertes_df.columns = kombiniertes_df.columns.str.replace('\n', ' ',
41     regex=True).str.strip()
42
43 # Überprüfen der bereinigten Spaltennamen
44 print("Bereinigte Spaltennamen:")
45 print(kombiniertes_df.columns.tolist())
46
47 # Filtern der Unterbezirke (Ausschließen der Oberbezirke und nicht
48     zugeordneten Einträge)
49 unterbezirke_df = kombiniertes_df[~kombiniertes_df['Bezeichnung (
50     Bezirksregion)'].isin(oberbezirke)]
51 print(f"Anzahl der Unterbezirke nach Filter: {unterbezirke_df.shape[0]}")
52
53 # Weitere Filterung: Entfernen von nicht zugeordneten Zeilen
54 # Angenommen, nicht zugeordnete Zeilen enthalten bestimmte Schlüsselwörter
55 nicht_zugeordnet_keywords = ['nicht zuzuordnen', 'Berlin (PKS gesamt)']
56 unterbezirke_df = unterbezirke_df[~unterbezirke_df['Bezeichnung (
57     Bezirksregion)'].str.contains('|'.join(nicht_zugeordnet_keywords), case=
58     False, na=False)]
59 print(f"Anzahl der Unterbezirke nach Entfernen nicht zugeordneter Zeilen: {
60     unterbezirke_df.shape[0]}")

```

```

56 # Sicherstellen, dass die Spalte 'Straftaten -insgesamt-' vorhanden ist
57 straftaten_spalte = 'Straftaten -insgesamt-'
58 if straftaten_spalte not in unterbezirke_df.columns:
59     raise ValueError(f"Die Spalte '{straftaten_spalte}' wurde nicht gefunden.
        Bitte überprüfen Sie die Spaltennamen.")
60
61 # Gruppieren nach Unterbezirk und Summieren der Straftaten
62 aggregiertes_df = unterbezirke_df.groupby('Bezeichnung (Bezirksregion)')[
    straftaten_spalte].sum().reset_index()
63
64 # Sortieren nach der Anzahl der Straftaten in absteigender Reihenfolge
65 aggregiertes_df = aggregiertes_df.sort_values(by=straftaten_spalte, ascending
    =False)
66
67 # Auswahl der Top 10 Unterbezirke mit den meisten Straftaten
68 top10_df = aggregiertes_df.head(10)
69
70 # Finale Darstellung mit nur den gewünschten Spalten
71 final_df = top10_df[['Bezeichnung (Bezirksregion)', straftaten_spalte]]
72
73 # Ausgabe des finalen DataFrames
74 print("Top 10 Unterbezirke mit den meisten Straftaten:")
75 print(final_df)
76
77 # Optional: Speichern des finalen DataFrames in eine neue Excel-Datei
78 final_df.to_excel('Top10_Unterbezirke_Straftaten.xlsx', index=False)
79 print("Das finale DataFrame wurde in 'Top10_Unterbezirke_Straftaten.xlsx'
    gespeichert.")

```

Listing 89: Skript Testfall 6 Prompt 2 Ausführung 3

```

1 import pandas as pd
2
3 # Dateipfad zur Excel-Datei
4 file_path = 'Fallzahlen&HZ 2014-2023.xlsx'
5
6 # Liste der Oberbezirke
7 oberbezirke = [
8     'Mitte',
9     'Friedrichshain-Kreuzberg',
10    'Pankow',
11    'Charlottenburg-Wilmersdorf',
12    'Spandau',
13    'Steglitz-Zehlendorf',
14    'Tempelhof-Schöneberg',
15    'Neukölln',
16    'Treptow-Köpenick',
17    'Marzahn-Hellersdorf',
18    'Lichtenberg',
19    'Reinickendorf'
20 ]
21
22 # Generiere die Liste der Sheet-Namen von 2014 bis 2023

```

```

23 sheet_names = [f'Fallzahlen_{year}' for year in range(2014, 2024)]
24
25 # Lese alle relevanten Sheets ein und speichere sie in einer Liste
26 dfs = []
27 for sheet in sheet_names:
28     try:
29         df = pd.read_excel(file_path, sheet_name=sheet)
30         dfs.append(df)
31     except ValueError:
32         print(f"Sheet {sheet} nicht gefunden in der Datei.")
33
34 # Kombiniere alle DataFrames zu einem einzigen DataFrame
35 combined_df = pd.concat(dfs, ignore_index=True)
36
37 # Bereinige die Spaltennamen: Entferne Zeilenumbrüche und führende/trailende
    Leerzeichen
38 combined_df.columns = combined_df.columns.str.replace('\n', ' ').str.strip()
39
40 # Identifiziere die Spalte für "Straftaten -insgesamt-"
41 # Diese Annahme basiert auf den bereitgestellten Beispieldaten
42 straftaten_spalte = 'Straftaten -insgesamt-'
43
44 if straftaten_spalte not in combined_df.columns:
45     # Falls die Spalte nicht exakt gefunden wird, suche ähnlich benannte
        Spalten
46     mögliche_spalten = [col for col in combined_df.columns if 'Straftaten' in
        col and 'insgesamt' in col.lower()]
47     if mögliche_spalten:
48         straftaten_spalte = mögliche_spalten[0]
49     else:
50         raise ValueError('Die Spalte für "Straftaten -insgesamt-" wurde nicht
        gefunden.')
```

```

51
52 # Filtere nur die Unterbezirke:
53 # - Schließe die Oberbezirke aus
54 # - Schließe Zeilen aus, die "nicht zuzuordnen" enthalten
55 unterbezirke_df = combined_df[
56     (~combined_df['Bezeichnung (Bezirksregion)'].isin(oberbezirke)) &
57     (~combined_df['Bezeichnung (Bezirksregion)'].str.contains('nicht
        zuzuordnen', case=False, na=False))
58 ]
59
60 # Stelle sicher, dass die Straftaten-Spalte numerisch ist
61 unterbezirke_df[straftaten_spalte] = pd.to_numeric(unterbezirke_df[
        straftaten_spalte], errors='coerce').fillna(0)
62
63 # Aggregiere die Straftaten über alle Jahre hinweg pro Unterbezirk
64 aggregated_df = unterbezirke_df.groupby('Bezeichnung (Bezirksregion)')[
        straftaten_spalte].sum().reset_index()
65
66 # Ermittle die Top 10 Unterbezirke mit den meisten Straftaten
67 top10_df = aggregated_df.sort_values(by=straftaten_spalte, ascending=False).
```



```

        head(10)
68
69 # Benenne die Spalten für den finalen DataFrame um (optional)
70 final_df = top10_df.rename(columns={
71     'Bezeichnung (Bezirksregion)': 'Unterbezirk',
72     'straftaten_spalte': 'Straftaten_Gesamt'
73 })
74
75 # Optional: Setze die Spalten in eine bestimmte Reihenfolge
76 final_df = final_df[['Unterbezirk', 'Straftaten_Gesamt']]
77
78 # Ausgabe des finalen DataFrames
79 print(final_df)
80
81 # Optional: Speichere das Ergebnis in eine neue Excel-Datei
82 # final_df.to_excel('Top10_Unterbezirke_Straftaten.xlsx', index=False)

```

Listing 90: Skript Testfall 6 Prompt 2 Ausführung 4

```

1 import pandas as pd
2
3 # Pfad zur Excel-Datei
4 excel_datei = 'Fallzahlen&HZ 2014-2023.xlsx'
5
6 # Liste der Oberbezirke
7 oberbezirke = [
8     'Mitte', 'Friedrichshain-Kreuzberg', 'Pankow', 'Charlottenburg-
9     Wilmersdorf',
10    'Spandau', 'Steglitz-Zehlendorf', 'Tempelhof-Schöneberg', 'Neukölln',
11    'Treptow-Köpenick', 'Marzahn-Hellersdorf', 'Lichtenberg', 'Reinickendorf'
12 ]
13
14 # Liste der relevanten Sheet-Namen
15 sheet_namen = [f'Fallzahlen_{jahr}' for jahr in range(2014, 2024)]
16
17 # Alle relevanten Sheets einlesen und in eine Liste von DataFrames speichern
18 daten_list = []
19 for sheet in sheet_namen:
20     try:
21         df = pd.read_excel(excel_datei, sheet_name=sheet)
22         daten_list.append(df)
23     except Exception as e:
24         print(f"Fehler beim Lesen des Sheets {sheet}: {e}")
25
26 # Alle DataFrames zu einem einzigen DataFrame zusammenführen
27 gesamt_df = pd.concat(daten_list, ignore_index=True)
28
29 # Anzeige der Spaltennamen zur Überprüfung (optional)
30 # print(gesamt_df.columns)
31
32 # Filtern der Unterbezirke:
33 # - Ausschließen der Oberbezirke
34 # - Ausschließen von nicht zugeordneten Zeilen

```

```

34 filtered_df = gesamtdf[
35     (~gesamtdf['Bezeichnung (Bezirksregion)'].isin(oberbezirke)) &
36     (~gesamtdf['Bezeichnung (Bezirksregion)'].str.contains('nicht zuzuordnen', case=False, na=False))
37 ]
38
39 # Sicherstellen, dass die 'Straftaten -insgesamt-' Spalte numerisch ist
40 filtered_df['Straftaten -insgesamt-'] = pd.to_numeric(filtered_df['Straftaten -insgesamt-'], errors='coerce')
41
42 # Entfernen von Zeilen mit NaN in 'Straftaten -insgesamt-'
43 filtered_df = filtered_df.dropna(subset=['Straftaten -insgesamt-'])
44
45 # Aggregieren der Straftaten pro Unterbezirk
46 aggregiert_df = filtered_df.groupby('Bezeichnung (Bezirksregion)')['Straftaten -insgesamt-'].sum().reset_index()
47
48 # Top 10 Unterbezirke mit den meisten Straftaten
49 top_10_df = aggregiert_df.sort_values(by='Straftaten -insgesamt-', ascending=False).head(10)
50
51 # Optional: Zur besseren Darstellung sortieren
52 top_10_df = top_10_df.sort_values(by='Straftaten -insgesamt-', ascending=False)
53
54 # Finale Ausgabe
55 print("Top 10 Unterbezirke mit den meisten Straftaten:")
56 print(top_10_df)
57
58 # Optional: Speichern des Ergebnisses in eine neue Excel-Datei
59 # top_10_df.to_excel('Top_10_Unterbezirke_Straftaten.xlsx', index=False)

```

Listing 91: Skript Testfall 6 Prompt 2 Ausführung 5

```

1 import pandas as pd
2
3 # 1. Definiere den Dateinamen
4 excel_file = 'Fallzahlen&HZ 2014-2023.xlsx'
5
6 # 2. Liste der Oberbezirke
7 oberbezirke = [
8     'Mitte', 'Friedrichshain-Kreuzberg', 'Pankow', 'Charlottenburg-
9     Wilmersdorf',
10    'Spandau', 'Steglitz-Zehlendorf', 'Tempelhof-Schöneberg', 'Neukölln',
11    'Treptow-Köpenick', 'Marzahn-Hellersdorf', 'Lichtenberg', 'Reinickendorf'
12 ]
13
14 # 3. Lese alle Sheets ein und speichere sie in einer Liste von DataFrames
15 # Annahme: Die Sheets heißen 'Fallzahlen_2014' bis 'Fallzahlen_2023'
16 jahre = range(2014, 2024)
17 dataframes = []
18
19 for jahr in jahre:

```

```

19     sheet_name = f'Fallzahlen_{jahr}'
20     try:
21         df = pd.read_excel(excel_file, sheet_name=sheet_name)
22         df['Jahr'] = jahr # Füge eine Spalte für das Jahr hinzu
23         dataframes.append(df)
24     except Exception as e:
25         print(f"Fehler beim Lesen des Sheets '{sheet_name}': {e}")
26
27 # 4. Füge alle DataFrames zu einem einzigen DataFrame zusammen
28 gesamt_df = pd.concat(dataframes, ignore_index=True)
29
30 # 5. Entferne Zeilen, die Oberbezirke, 'Berlin (PKS gesamt)' oder nicht
    zuzuordnende Einträge enthalten
31 # Definiere die Namen der nicht zuzuordnenden Einträge
32 nicht_zugeordnet = [
33     'Bezirk (Rd), nicht zuzuordnen',
34     'Stadtgebiet Berlin, nicht zuzuordnen',
35     'Berlin (PKS gesamt)'
36 ]
37
38 # Filtere die DataFrame, um nur Unterbezirke zu behalten
39 unterbezirk_df = gesamt_df[~gesamt_df['Bezeichnung (Bezirksregion)'].isin(
    oberbezirke + nicht_zugeordnet)]
40
41 # 6. Summiere die Straftaten über alle Jahre hinweg für jeden Unterbezirk
42 # Achte auf den genauen Spaltennamen, möglicherweise enthält er einen
    Zeilenumbruch
43 # Daher benutze hier eine Rohzeichenkette oder ersetze Zeilenumbrüche
44 straftaten_spalte = 'Straftaten \n-insgesamt-' # Anpassung basierend auf dem
    Beispiel
45
46 # Überprüfe, ob die Spalte existiert
47 if straftaten_spalte not in unterbezirk_df.columns:
48     # Alternativ Zeilenumbrüche entfernen und erneut versuchen
49     unterbezirk_df.columns = unterbezirk_df.columns.str.replace('\n', ' ',
    regex=True)
50     straftaten_spalte = 'Straftaten -insgesamt-'
51     if straftaten_spalte not in unterbezirk_df.columns:
52         raise ValueError(f"Die Spalte '{straftaten_spalte}' wurde nicht
    gefunden.")
53
54 # Gruppiere nach Unterbezirk und summiere die Straftaten
55 aggregierte_straftaten = unterbezirk_df.groupby('Bezeichnung (Bezirksregion)')
    [straftaten_spalte].sum()
56
57 # 7. Sortiere die Unterbezirke absteigend nach den Straftaten
58 sortierte_straftaten = aggregierte_straftaten.sort_values(ascending=False)
59
60 # 8. Wähle die Top 10 Unterbezirke mit den höchsten Straftaten
61 top_10_unterbezirke = sortierte_straftaten.head(10)
62
63 # 9. Erstelle einen neuen DataFrame mit den Top 10 Unterbezirken und ihren

```

```

        Straftaten
64 final_df = top_10_unterbezirke.reset_index()
65 final_df.columns = ['Bezeichnung (Bezirksregion)', 'Straftaten -insgesamt-']
66
67 # 10. Gib den finalen DataFrame aus
68 print(final_df)

```

Listing 92: Skript Testfall 6 Prompt 3 Ausführung 1

```

1 import pandas as pd
2
3 # Schritt 1: Definieren Sie den Dateipfad und die relevanten Sheetnamen
4 excel_datei = 'Fallzahlen&HZ 2014-2023.xlsx'
5 jahre = range(2014, 2024) # 2014 bis 2023 inclusive
6 sheet_namen = [f'Fallzahlen_{jahr}' for jahr in jahre]
7
8 # Schritt 2: Lesen Sie alle Sheets ein und speichern Sie sie in separaten
   DataFrames
9 dataframes = {}
10 for sheet in sheet_namen:
11     try:
12         df = pd.read_excel(excel_datei, sheet_name=sheet)
13         dataframes[sheet] = df
14         print(f"Sheet '{sheet}' erfolgreich gelesen.")
15     except Exception as e:
16         print(f"Fehler beim Lesen des Sheets '{sheet}': {e}")
17
18 # Prüfen, ob alle Sheets erfolgreich gelesen wurden
19 if len(dataframes) != len(sheet_namen):
20     print("Nicht alle Sheets wurden erfolgreich gelesen. Bitte überprüfen Sie
       die Sheetnamen und die Excel-Datei.")
21     exit()
22
23 # Schritt 3: Fügen Sie alle DataFrames zu einem einzigen DataFrame zusammen
24 gesamt_df = pd.concat(dataframes.values(), ignore_index=True)
25 print("Alle Sheets wurden erfolgreich zu einem einzigen DataFrame zusammengef
       ührt.")
26
27 # Schritt 4: Definieren Sie die Oberbezirke
28 oberbezirke = [
29     'Mitte',
30     'Friedrichshain-Kreuzberg',
31     'Pankow',
32     'Charlottenburg-Wilmersdorf',
33     'Spandau',
34     'Steglitz-Zehlendorf',
35     'Tempelhof-Schöneberg',
36     'Neukölln',
37     'Treptow-Köpenick',
38     'Marzahn-Hellersdorf',
39     'Lichtenberg',
40     'Reinickendorf'
41 ]

```

```

42
43 # Schritt 5: Definieren Sie die Ausschlusskriterien
44 ausschlusskriterien = oberbezirke + [
45     'Berlin (PKS gesamt)',
46     'Stadt Berlin, nicht zuzuordnen',
47     'Bezirk (Rd), nicht zuzuordnen',
48     'Stadtgebiet Berlin, nicht zuzuordnen'
49 ]
50
51 # Schritt 6: Filtern Sie die DataFrame, um nur Unterbezirke zu behalten
52 unterbezirke_df = gesamt_df[~gesamt_df['Bezeichnung (Bezirksregion)'].isin(
53     ausschlusskriterien)]
54 print(f"Anzahl der verbleibenden Unterbezirke: {unterbezirke_df['Bezeichnung
55     (Bezirksregion)'].nunique()}")
56
57 # Schritt 7: Summieren Sie die Straftaten pro Unterbezirk über alle Jahre
58     hinweg
59 # Stellen Sie sicher, dass die Spalte 'Straftaten \n-insgesamt-' numerisch
60     ist
61 unterbezirke_df['Straftaten \n-insgesamt-'] = pd.to_numeric(unterbezirke_df['
62     Straftaten \n-insgesamt-'], errors='coerce')
63
64 # Gruppieren nach 'Bezeichnung (Bezirksregion)' und summieren
65 straftaten_summe = unterbezirke_df.groupby('Bezeichnung (Bezirksregion)')['
66     Straftaten \n-insgesamt-'].sum().reset_index()
67
68 # Schritt 8: Sortieren Sie die Unterbezirke nach der aggregierten Anzahl an
69     Straftaten absteigend
70 straftaten_sortiert = straftaten_summe.sort_values(by='Straftaten \n-
71     insgesamt-', ascending=False)
72
73 # Schritt 9: Wählen Sie die Top 10 Unterbezirke mit den höchsten
74     Gesamtstraftaten aus
75 top10_unterbezirke = straftaten_sortiert.head(10)
76
77 # Schritt 10: Erstellen Sie den finalen DataFrame mit den gewünschten Spalten
78 finaler_df = top10_unterbezirke[['Bezeichnung (Bezirksregion)', 'Straftaten \
79     n-insgesamt-']]
80
81 # Schritt 11: Ausgabe des finalen DataFrames
82 print("\nTop 10 Unterbezirke mit den höchsten Gesamtstraftaten:")
83 print(finaler_df.to_string(index=False))

```

Listing 93: Skript Testfall 6 Prompt 3 Ausführung 2

```

1 import pandas as pd
2
3 # 1. Alle Sheets einlesen und in separaten DataFrames speichern
4 excel_datei = 'Fallzahlen&HZ 2014-2023.xlsx'
5 sheet_namen = [f'Fallzahlen_{jahr}' for jahr in range(2014, 2024)] # 2014
6     bis 2023
7
8 # Dictionary zur Speicherung der DataFrames

```

```

8 dataframes = {}
9
10 for sheet in sheet_namen:
11     try:
12         df = pd.read_excel(excel_datei, sheet_name=sheet)
13         dataframes[sheet] = df
14     except Exception as e:
15         print(f"Fehler beim Lesen des Sheets {sheet}: {e}")
16
17 # 2. Alle DataFrames zu einem einzigen zusammenführen
18 gesamtdf = pd.concat(dataframes.values(), ignore_index=True)
19
20 # 3. Oberbezirke definieren
21 oberbezirke = [
22     'Mitte', 'Friedrichshain-Kreuzberg', 'Pankow', 'Charlottenburg-
23     Wilmersdorf',
24     'Spandau', 'Steglitz-Zehlendorf', 'Tempelhof-Schöneberg', 'Neukölln',
25     'Treptow-Köpenick', 'Marzahn-Hellersdorf', 'Lichtenberg', 'Reinickendorf'
26 ]
27
28 # 4. Zeilen entfernen, die Oberbezirke, 'Berlin (PKS gesamt)', 'Stadt Berlin,
29     nicht zuzuordnen' oder ähnliche enthalten
30 begrenzende_werte = oberbezirke + ['Berlin (PKS gesamt)', 'Stadtgebiet Berlin
31     , nicht zuzuordnen', 'Bezirk (Rd), nicht zuzuordnen']
32
33 # Annahme: Spaltenname genau 'Bezeichnung (Bezirksregion)'
34 unterbezirk_df = gesamtdf[~gesamtdf['Bezeichnung (Bezirksregion)'].isin(
35     begrenzende_werte)].copy()
36
37 # 5. Straftaten insgesamt über alle Jahre summieren
38 # Beachten Sie den genauen Spaltennamen, eventuell mit Zeilenumbruch und
39     Bindestrich
40 # Um den Spaltennamen sicher zu erfassen, können wir nach einer Teilü
41     bereinstimmung suchen
42 # Alternativ den exakten Spaltennamen verwenden, wie im Beispiel 'Straftaten
43     \n-insgesamt-'
44
45 # Finden des exakten Spaltennamens
46 spaltennamen = unterbezirk_df.columns
47 straftaten_spalte = [spalte for spalte in spaltennamen if 'Straftaten' in
48     spalte and 'insgesamt' in spalte]
49 if not straftaten_spalte:
50     raise ValueError("Die Spalte für 'Straftaten insgesamt' wurde nicht
51     gefunden.")
52 straftaten_spalte = straftaten_spalte[0]
53
54 # Gruppieren nach Unterbezirk und Summieren der Straftaten
55 aggregiert = unterbezirk_df.groupby('Bezeichnung (Bezirksregion)')[
56     straftaten_spalte].sum().reset_index()
57
58 # 6. Sortieren der Unterbezirke absteigend nach Straftaten
59 aggregiert_sortiert = aggregiert.sort_values(by=straftaten_spalte, ascending=

```

```

False)
50
51 # 7. Top 10 Unterbezirke auswählen
52 top_10 = aggregiert_sortiert.head(10)
53
54 # 8. Neuer DataFrame mit den gewünschten Spalten
55 final_df = top_10[['Bezeichnung (Bezirksregion)', straftaten_spalte]].copy()
56
57 # Optional: Spalten umbenennen für Klarheit
58 final_df.rename(columns={
59     'Bezeichnung (Bezirksregion)': 'Unterbezirk',
60     straftaten_spalte: 'Gesamte Straftaten'
61 }, inplace=True)
62
63 # 9. Finalen DataFrame ausgeben
64 print(final_df)

```

Listing 94: Skript Testfall 6 Prompt 3 Ausführung 3

```

1 import pandas as pd
2
3 # 1. Definiere den Dateinamen und die zu lesenden Sheets
4 excel_file = 'Fallzahlen&HZ 2014-2023.xlsx'
5 jahre = list(range(2014, 2024)) # 2014 bis 2023
6 sheet_names = [f'Fallzahlen_{jahr}' for jahr in jahre]
7
8 # 2. Lese alle Sheets ein und speichere sie in einer Liste von DataFrames
9 dataframes = []
10 for sheet in sheet_names:
11     df = pd.read_excel(excel_file, sheet_name=sheet)
12     dataframes.append(df)
13
14 # 3. Füge alle DataFrames zu einem einheitlichen DataFrame zusammen
15 all_data = pd.concat(dataframes, ignore_index=True)
16
17 # 4. Definiere die Oberbezirke
18 oberbezirke = [
19     'Mitte', 'Friedrichshain-Kreuzberg', 'Pankow', 'Charlottenburg-
20     Wilmersdorf',
21     'Spandau', 'Steglitz-Zehlendorf', 'Tempelhof-Schöneberg', 'Neukölln',
22     'Treptow-Köpenick', 'Marzahn-Hellersdorf', 'Lichtenberg', 'Reinickendorf'
23 ]
24
25 # Zusätzliche nicht zuzuordnende Einträge
26 nicht_zuordenbar = [
27     'Berlin (PKS gesamt)',
28     'Bezirk (Rd), nicht zuzuordnen',
29     'Stadtgebiet Berlin, nicht zuzuordnen'
30 ]
31
32 # 5. Entferne alle Zeilen, die Oberbezirke oder nicht zuordenbare Einträge
33     enthalten
34 filtered_data = all_data[~all_data['Bezeichnung (Bezirksregion)'].isin(

```

```

    oberbezirke + nicht_zuordenbar]])
33
34 # 6. Entferne weitere allgemeine Einträge, falls vorhanden (optional)
35 # Falls es noch andere allgemeine Einträge gibt, können diese hier hinzugefü
    gt werden
36 # Beispielsweise: 'Gesamt Berlin' oder ähnliche
37
38 # 7. Aggregiere die Straftaten pro Unterbezirk über alle Jahre hinweg
39 # Zuerst sicherstellen, dass die Spalte korrekt benannt ist
40 # Manchmal können Leerzeichen oder Zeilenumbrüche Probleme verursachen
41
42 # Liste der möglichen Spaltennamen für 'Straftaten -insgesamt-'
43 straftaten_spalten = [
44     'Straftaten \n-insgesamt-',
45     'Straftaten -insgesamt-',
46     'Straftaten - insgesamt -' # Falls es Varianten gibt
47 ]
48
49 # Finde die existierende Spalte
50 straftaten_spalte = None
51 for col in straftaten_spalten:
52     if col in filtered_data.columns:
53         straftaten_spalte = col
54         break
55
56 if not straftaten_spalte:
57     raise ValueError("Die Spalte für 'Straftaten insgesamt' wurde nicht
    gefunden.")
58
59 # Gruppiere nach 'Bezeichnung (Bezirksregion)' und summiere die Straftaten
60 aggregated = filtered_data.groupby('Bezeichnung (Bezirksregion)')[
    straftaten_spalte].sum().reset_index()
61
62 # 8. Sortiere die Unterbezirke absteigend nach der aggregierten Anzahl an
    Straftaten
63 aggregated_sorted = aggregated.sort_values(by=straftaten_spalte, ascending=
    False)
64
65 # 9. Wähle die Top 10 Unterbezirke mit den höchsten Gesamtstraftaten aus
66 top_10 = aggregated_sorted.head(10)
67
68 # 10. Erstelle den finalen DataFrame mit den gewünschten Spalten
69 final_df = top_10[['Bezeichnung (Bezirksregion)', straftaten_spalte]].copy()
70 final_df = final_df.rename(columns={
71     'Bezeichnung (Bezirksregion)': 'Unterbezirk',
72     straftaten_spalte: 'Gesamtstraftaten'
73 })
74
75 # 11. Ausgabe des finalen DataFrames
76 print(final_df)

```

Listing 95: Skript Testfall 6 Prompt 3 Ausführung 4


```

1 import pandas as pd
2
3 # 1. Definiere den Pfad zur Excel-Datei
4 excel_datei = 'Fallzahlen&HZ 2014-2023.xlsx'
5
6 # 2. Erstelle eine Liste der Sheet-Namen von 'Fallzahlen_2014' bis '
   Fallzahlen_2023'
7 jahre = range(2014, 2024) # 2024 ist exklusiv, also bis 2023
8 sheet_namen = [f'Fallzahlen_{jahr}' for jahr in jahre]
9
10 # 3. Lese alle Sheets ein und speichere jedes in einem separaten DataFrame
11 dataframes = []
12 for sheet in sheet_namen:
13     try:
14         df = pd.read_excel(excel_datei, sheet_name=sheet)
15         dataframes.append(df)
16         print(f"Sheet '{sheet}' erfolgreich eingelesen.")
17     except Exception as e:
18         print(f"Fehler beim Einlesen von Sheet '{sheet}': {e}")
19
20 # 4. Füge alle DataFrames zu einem einzigen zusammen
21 kombinierter_df = pd.concat(dataframes, ignore_index=True)
22 print("Alle Sheets erfolgreich zu einem DataFrame kombiniert.")
23
24 # 5. Definiere die Liste der Oberbezirke
25 oberbezirke = [
26     'Mitte',
27     'Friedrichshain-Kreuzberg',
28     'Pankow',
29     'Charlottenburg-Wilmersdorf',
30     'Spandau',
31     'Steglitz-Zehlendorf',
32     'Tempelhof-Schöneberg',
33     'Neukölln',
34     'Treptow-Köpenick',
35     'Marzahn-Hellersdorf',
36     'Lichtenberg',
37     'Reinickendorf'
38 ]
39
40 # 6. Definiere zusätzliche auszuschließende Einträge
41 ausschluss_liste = oberbezirke + [
42     'Berlin (PKS gesamt)',
43     'Stadtgebiet Berlin, nicht zuzuordnen',
44     'Bezirk (Rd), nicht zuzuordnen'
45 ]
46
47 # 7. Entferne alle Zeilen, die in der Ausschlussliste enthalten sind
48 unterbezirke_df = kombinierter_df[~kombinierter_df['Bezeichnung (
   Bezirksregion)'].isin(ausschluss_liste)]
49 print(f"Anzahl der verbleibenden Unterbezirke: {unterbezirke_df['Bezeichnung
   (Bezirksregion)'].unique()}")

```

```

50
51 # 8. Stelle sicher, dass die Spalte 'Straftaten \n-insgesamt-' numerisch ist
52 unterbezirke_df['Straftaten \n-insgesamt-'] = pd.to_numeric(unterbezirke_df['
    Straftaten \n-insgesamt-'], errors='coerce')
53
54 # 9. Entferne mögliche NaN-Werte in der Straftaten-Spalte
55 unterbezirke_df = unterbezirke_df.dropna(subset=['Straftaten \n-insgesamt-'])
56
57 # 10. Summiere die Straftaten pro Unterbezirk über alle Jahre hinweg
58 aggregierte_straftaten = unterbezirke_df.groupby('Bezeichnung (Bezirksregion)
    ')[ 'Straftaten \n-insgesamt-'].sum()
59
60 # 11. Sortiere die Unterbezirke absteigend nach der aggregierten Anzahl an
    Straftaten
61 sortierte_straftaten = aggregierte_straftaten.sort_values(ascending=False)
62
63 # 12. Wähle die Top 10 Unterbezirke mit den höchsten Gesamtstraftaten aus
64 top_10_unterbezirke = sortierte_straftaten.head(10)
65
66 # 13. Erstelle einen neuen DataFrame mit den gewünschten Spalten
67 finaler_df = top_10_unterbezirke.reset_index()
68 finaler_df.columns = ['Bezeichnung (Bezirksregion)', 'Straftaten -insgesamt-'
    ]
69
70 # 14. Gib den finalen DataFrame aus
71 print("\nTop 10 Unterbezirke mit den höchsten Gesamtstraftaten:")
72 print(finaler_df)

```

Listing 96: Skript Testfall 6 Prompt 3 Ausführung 5

11.3.7 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 97: 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 98: 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 99: 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 100: Manuelles Skript Testfall 4

```

1 import pandas as pd
2
3 # read excel
4 df = pd.read_excel("Fallzahlen&HZ 2014-2023.xlsx", sheet_name="
    Fallzahlen_2023", usecols=["Bezeichnung (Bezirksregion)", "Raub"])
5
6 # remove last two rows
7 df = df.iloc[:-2]
8
9 # list Oberbezirke
10 oberbezirke = {
11     "Mitte", "Friedrichshain-Kreuzberg", "Pankow", "Charlottenburg -
    Wilmersdorf",
12     "Spandau", "Steglitz-Zehlendorf", "Tempelhof-Schöneberg", "Neukölln",

```

```

13     "Treptow-Köpenick", "Marzahn-Hellersdorf", "Lichtenberg", "Reinickendorf"
14 }
15
16 # put oberbezirk in oberbezirk column
17 df["Oberbezirk"] = df["Bezeichnung (Bezirksregion)"].where(df["Bezeichnung (
    Bezirksregion)"].isin(oberbezirke)).ffill()
18
19 # keep only unterbezirke
20 df = df[~df["Bezeichnung (Bezirksregion)"].isin(oberbezirke)].copy()
21
22 # convert Raub to numeric
23 df["Raub"] = pd.to_numeric(df["Raub"], errors="coerce")
24
25 # get top raub unterbezirke
26 top_raub_unterbezirke = df.loc[df.groupby("Oberbezirk")["Raub"].idxmax(), ["
    Oberbezirk", "Bezeichnung (Bezirksregion)", "Raub"]]
27
28 # rename columns
29 top_raub_unterbezirke.rename(columns={"Bezeichnung (Bezirksregion)": "
    Unterbezirk"}, inplace=True)
30
31 print(top_raub_unterbezirke)

```

Listing 101: Manuelles Skript Testfall 5

```

1 import pandas as pd
2
3 file = 'Fallzahlen&HZ 2014-2023.xlsx'
4
5 # Sheets from 2014 to 2023
6 sheets = [f'Fallzahlen_{year}' for year in range(2014, 2024)]
7 dfs = []
8
9 # Lines to ignore
10 ignore_lines = [
11     "Mitte", "Friedrichshain-Kreuzberg", "Pankow", "Charlottenburg-
    Wilmersdorf",
12     "Spandau", "Steglitz-Zehlendorf", "Tempelhof-Schöneberg", "Neukölln",
13     "Treptow-Köpenick", "Marzahn-Hellersdorf", "Lichtenberg", "Reinickendorf"
14     ,
15     "Berlin (PKS gesamt)", "Stadtgebiet Berlin, nicht zuzuordnen"
16 ]
17
18 # Read the needed sheets in loop, use only the required columns and add to df
    list
19 with pd.ExcelFile(file) as xls:
20     for sheet in sheets:
21         if sheet in xls.sheet_names:
22             df = pd.read_excel(xls, sheet_name=sheet, usecols=["Bezeichnung (
                Bezirksregion)", "Straftaten \n-insgesamt-"])
23             dfs.append(df)
24
25 # Combine all dataframes

```

```

25 combined_df = pd.concat(dfs, ignore_index=True)
26 # Check if the column is numeric
27 combined_df["Straftaten \n-insgesamt-"] = pd.to_numeric(combined_df["
    Straftaten \n-insgesamt-"], errors='coerce')
28
29 # Filter out the ignore lines
30 filtered_df = combined_df[~combined_df['Bezeichnung (Bezirksregion)'].isin(
    ignore_lines)]
31
32 # Group df
33 grouped_df = filtered_df.groupby('Bezeichnung (Bezirksregion)', as_index=
    False)['Straftaten \n-insgesamt-'].sum()
34
35 # Sort df
36 sorted_df = grouped_df.sort_values(by="Straftaten \n-insgesamt-", ascending=
    False)
37
38 # Get top 10
39 top_10 = sorted_df.head(10)
40
41 print(top_10)

```

Listing 102: Manuelles Skript Testfall 6

11.3.8 Komplette Antworten der API Requests Testfall 1

```

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=
26     False)
27
28     # Zurücksetzen des Index (optional)
29     df_sortiert.reset_index(drop=True, inplace=True)
30
31     # Anzeige der sortierten DataFrame
32     print("\nSortierter DataFrame nach 'Straftaten_insgesamt':")
33     print(df_sortiert)
34
35     # Optional: Speichern des sortierten DataFrames in eine neue Excel-
36     Datei
37     sortierte_datei_pfad = 'Fallzahlen_sortiert.xlsx'
38     df_sortiert.to_excel(sortierte_datei_pfad, index=False)
39     print(f"\nDer sortierte DataFrame wurde in '{sortierte_datei_pfad}'
40     gespeichert.")
41
42 except FileNotFoundError:
43     print(f"Die Datei '{datei_pfad}' wurde nicht gefunden.")
44 except ValueError as ve:
45     print(f"Fehler beim Einlesen des Sheets: {ve}")
46 except Exception as e:
47     print(f"Ein unerwarteter Fehler ist aufgetreten: {e}")
48
49 '''
50
51 3. **Erklärung der Schritte**:
52
53 - **Importieren der Bibliotheken**:
54     '''python
55     import pandas as pd
56     '''
57     'pandas' ist eine leistungsstarke Bibliothek zur Datenanalyse in Python.
58
59 - **Einlesen der Excel-Datei**:
60     '''python
61     df = pd.read_excel(datei_pfad, sheet_name='Fallzahlen_2023')
62     '''
63     Hier wird das spezifische Sheet 'Fallzahlen_2023' aus der Excel-Datei
64     eingelesen.
65
66 - **Sortieren des DataFrames**:
67     '''python
68     df_sortiert = df.sort_values(by='Straftaten_insgesamt', ascending=False)
69     '''
70     Der DataFrame wird basierend auf der Spalte 'Straftaten_insgesamt' in
71     absteigender Reihenfolge sortiert.
72
73 - **Zurücksetzen des Index**:
74     '''python
75     df_sortiert.reset_index(drop=True, inplace=True)
76     '''

```

```

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

```

Listing 103: API Response Testfall 1 Prompt 1 Ausführung 1

```

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. Wir
    werden dafür die **pandas** Bibliothek verwenden, die sich hervorragend fü
    r die Datenanalyse eignet. Falls du **pandas** und **openpyxl** noch nicht
    installiert hast, kannst du dies mit 'pip' tun.
2
3 ### Schritt 1: Installation der benötigten Bibliotheken
4
5 Öffne dein Terminal oder die Eingabeaufforderung und führe die folgenden
    Befehle aus:
6
7     '''bash
8     pip install pandas openpyxl
9     '''
10
11 ### Schritt 2: Python-Skript erstellen
12
13 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
51     arbeiten.
52
53 2. **Pfad und Sheet-Name festlegen:**
54     '''python
55     excel_datei = 'Fallzahlen.xlsx'
56     sheet_name = 'Fallzahlen_2023'
57     '''
58     Stelle sicher, dass sich die Excel-Datei im gleichen Verzeichnis wie dein
59     Python-Skript befindet. Andernfalls musst du den vollständigen Pfad
60     angeben.
61
62 3. **Einlesen der Excel-Datei:**
63     '''python
64     df = pd.read_excel(excel_datei, sheet_name=sheet_name)
65     '''
66     Dies liest das angegebene Sheet in einen pandas DataFrame ein.
67
68 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
       absteigender Reihenfolge sortiert. Wenn du in aufsteigender Reihenfolge
       sortieren möchtest, setze 'ascending=True'.
77
78 6. **Anzeigen der sortierten Daten (optional):**
79     '''python
80     print("\nSortierte Daten nach 'Straftaten_insgesamt':")
81     print(df_sortiert.head())
82     '''
83     Dadurch kannst du die oberen Einträge der sortierten Daten überprüfen.
84
85 7. **Speichern der sortierten Daten in einer neuen Excel-Datei (optional):**
86     '''python
87     df_sortiert.to_excel('Fallzahlen_sortiert.xlsx', index=False)
88     '''
89     Dies speichert den sortierten DataFrame in einer neuen Excel-Datei namens
       "Fallzahlen_sortiert.xlsx". Der Parameter 'index=False' verhindert,
       dass der DataFrame-Index in die Excel-Datei geschrieben wird.
90
91 8. **Abschlussnachricht:**
92     '''python
93     print("\nDie sortierten Daten wurden in 'Fallzahlen_sortiert.xlsx'
94           gespeichert.")
95     '''
96
97 ### Ausführen des Skripts
98
99 Nachdem du das Skript erstellt hast, kannst du es ausführen, indem du im
       Terminal oder der Eingabeaufforderung in das Verzeichnis navigierst, in
       dem sich dein Skript befindet, und den folgenden Befehl eingibst:
100
101     '''bash
102     python sortiere_fallzahlen.py
103     '''
104
105 Nach der Ausführung solltest du eine neue Excel-Datei namens "
       Fallzahlen_sortiert.xlsx" in deinem Verzeichnis finden, die die
       sortierten Daten enthält.
106
107 ### Zusätzliche Hinweise
108
109 - **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 104: API 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
26 DataFrame nicht gefunden.")
27
28     # Sortieren des DataFrames nach 'Straftaten_insgesamt' in absteigender
29 Reihenfolge
30     df_sorted = df.sort_values(by='Straftaten_insgesamt', ascending=False).
31 reset_index(drop=True)
32     print("DataFrame erfolgreich nach 'Straftaten_insgesamt' sortiert.")
33
34     # Optional: Anzeige der sortierten Daten
35     print(df_sorted)
36
37     # Optional: Speichern des sortierten DataFrames in eine neue Excel-Datei
38     # df_sorted.to_excel('Fallzahlen_sorted.xlsx', index=False)
39     # print("Sortierte Daten wurden in 'Fallzahlen_sorted.xlsx' gespeichert
40     .")
41
42 except FileNotFoundError:
43     print(f"Die Datei '{excel_datei}' wurde nicht gefunden.")
44 except ValueError as ve:
45     print(f"Fehler: {ve}")
46 except Exception as e:
47     print(f"Ein unerwarteter Fehler ist aufgetreten: {e}")
48 '''
49
50 ### Erklärung des Skripts
51
52 1. **Importieren der benötigten Bibliothek:**
53     '''python
54     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'.
98     - Der 'index=False' Parameter stellt sicher, dass die Indizes nicht als
99       separate Spalte gespeichert werden.
100
101 8. **Fehlerbehandlung:**
102     - 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=[
    False, True])
117     '''
118   - Dies sortiert zuerst nach 'Straftaten_insgesamt' absteigend und dann nach
    'Raub' aufsteigend.
119
120 Wenn du weitere Anpassungen oder zusätzliche Funktionen benötigst, lass es
    mich wissen!

```

Listing 105: API 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):**
58     '''python
59     print(df_sortiert)
60     '''
61     Dies gibt den sortierten DataFrame im Terminal oder der Konsole aus.

```

```

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 106: API 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
60       im selben Verzeichnis wie das Skript befindet, reicht der Dateiname.
61
62 3. **Einlesen der Excel-Datei:**
63     - '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 107: API 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   ):
5   '''bash
6   pip install pandas openpyxl
7   '''
8
9 2. **Python-Skript**:
10
11   '''python
12   import pandas as pd
13
14   # Pfad zur Excel-Datei
15   excel_datei = 'Fallzahlen.xlsx'
16
17   # Name des Sheets
18   sheet_name = 'Fallzahlen_2023'
19
20   # LÖR-Schlüssel, die ausgeschlossen werden sollen
21   ausschluss_schluesel = [999900, 999999]

```

```

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_schluesel)]
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
60     Pandas ist eine leistungsstarke Bibliothek zur Datenmanipulation und -
61     analyse.
62
63 - **Einlesen der Excel-Datei**:
64     '''python
65     df = pd.read_excel(excel_datei, sheet_name=sheet_name)
66     '''
67
68     Liest das spezifizierte Sheet aus der Excel-Datei in ein DataFrame ein.
69
70 - **Filtern der unerwünschten Zeilen**:
71     '''python
72     df_filtered = df[~df['LOR-Schlüssel'].isin(ausschluss_schluesel)]
73     '''
74
75     Entfernt die Zeilen, bei denen der 'LOR-Schlüssel' entweder 999900 oder

```

```

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

```

Listing 108: API 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
15 '''
16
17 ### Python-Skript
18
19 '''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_schluesssel)
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=
  False).reset_index(drop=True)
80 '''
81 - Sortiert die gefilterten Daten nach 'Straftaten_insgesamt' in
  absteigender Reihenfolge ('ascending=False').
82 - 'reset_index(drop=True)' setzt den Index des DataFrames zurück, sodass
  er fortlaufend von 0 beginnt.
83
84 6. **Anzeige des Ergebnisses:**
85 '''python
86 print(df_sorted)
87 '''
88 - Gibt den sortierten DataFrame in der Konsole aus.
89
90 7. **Optional: Speichern in eine neue Excel-Datei:**
91 '''python
92 df_sorted.to_excel('Fallzahlen_sortiert.xlsx', index=False)
93 '''
94 - Speichert den sortierten DataFrame in eine neue Excel-Datei namens '
  Fallzahlen_sortiert.xlsx'.
95 - 'index=False' vermeidet, dass der DataFrame-Index in der Excel-Datei
  gespeichert wird.
96
97 ### Ausführen des Skripts
98
99 Speichern Sie das Skript in einer Datei, z.B. 'sortiere_fallzahlen.py', und f
  ühren Sie es in Ihrer Python-Umgebung aus:
100
101 '''bash
102 python sortiere_fallzahlen.py
103 '''
104 Nach der Ausführung sollten die sortierten Daten im Terminal angezeigt werden
  . Wenn Sie die Option zum Speichern aktiviert haben, finden Sie die

```

```

105     sortierten Daten auch in der neuen Excel-Datei 'Fallzahlen_sortiert.xlsx'.
106 ### Beispielausgabe
107
108 Nach dem Ausführen des Skripts sollte eine sortierte Tabelle angezeigt werden
109     , zum Beispiel:
110
111     '''
112     Sortierte Daten nach 'Straftaten_insgesamt':
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
129     wissen!

```

Listing 109: API Response Testfall 1 Prompt 2 Ausführung 2

```

1 Natürlich! Hier ist ein Python-Skript, das die beschriebenen Anforderungen
2     erfüllt. Das Skript verwendet die 'pandas'-Bibliothek, um die Excel-Datei
3     einzulesen, die Daten zu filtern, zu sortieren und in einem DataFrame zu
4     speichern.
5
6 Stellen Sie sicher, dass Sie die 'pandas'-Bibliothek installiert haben. Wenn
7     nicht, können Sie sie mit 'pip' installieren:
8
9     '''bash
10     pip install pandas
11     '''
12
13 Hier ist das vollständige Skript:
14
15     '''python
16     import pandas as pd
17
18     # Pfad zur Excel-Datei
19     dateipfad = 'Fallzahlen.xlsx'
20
21     # Name des Sheets
22     sheet_name = 'Fallzahlen_2023'
23
24     # 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 = [999900, 999999]
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     ausgeschlossene_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 110: API 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.
90
91 6. **Sortieren nach 'Straftaten_insgesamt':**
92     '''python
93     df_sortiert = df_gefiltert.sort_values(by='Straftaten_insgesamt',
94     ascending=False)

```

```

89     '''
90     Das DataFrame 'df_gefiltert' wird basierend auf der Spalte '
91     Straftaten_insgesamt' in absteigender Reihenfolge sortiert.
92
93 7. **Zurücksetzen des Index (optional):**
94     '''python
95     df_sortiert.reset_index(drop=True, inplace=True)
96     '''
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
106     Falls du das Ergebnis in eine neue Excel-Datei speichern möchtest, kannst
107     du diese Zeile entkommentieren.
108
109 ### Beispielausgabe
110
111 Das Skript enthält 'print'-Anweisungen, die die Daten in den verschiedenen
112 Schritten anzeigen. Dies ist nützlich für die Überprüfung, kann aber bei
113 großen Datensätzen entfernt oder angepasst werden.
114
115 '''plaintext
116 Ursprüngliche Daten:
117
118     LOR-Schlüssel      Bezirke  Straftaten_insgesamt  Raub  Stra
119     ßenraub  Körperverletzungen_insgesamt  ...  Branddelikte_insgesamt
120     Brandstiftung  Sachbeschädigung_insgesamt  Sachbeschädigung_Graffiti
121     Rauschgiftdelikte  Kieztaten
122
123 0      10000      Mitte      83611      884
124     521      8144  ...      420
125     137      6172      976
126     3306      19212
127
128 1      20000  Friedrichshain-Kreuzberg      55838      923
129     606      5502  ...      298
130     85      4906      801
131     4455      12724
132
133 ...      ...      ...      ...      ...
134     ...      ...      ...      ...
135     ...      ...      ...
136     ...      ...
137
138 Gefilterte Daten (ohne nicht zuzuordnende Bezirke):
139
140     LOR-Schlüssel      Bezirke  Straftaten_insgesamt  Raub  Stra
141     ßenraub  Körperverletzungen_insgesamt  ...  Branddelikte_insgesamt
142     Brandstiftung  Sachbeschädigung_insgesamt  Sachbeschädigung_Graffiti
143     Rauschgiftdelikte  Kieztaten
144
145 0      10000      Mitte      83611      884
146     521      8144  ...      420
147     137      6172      976
148     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 111: API 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
   korrekt eingelesen wurden
35   print("\nErste 5 Zeilen der Originaldaten:")
36   print(df.head())
37
38   # Filterkriterien: Ausschließen der LOR-Schlüssel 999900 und 999999
39   ausschluss_schluesel = [999900, 999999]
40   df_filtered = df[~df['LOR-Schlüssel'].isin(ausschluss_schluesel)]
41
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=
    False).reset_index(drop=True)
46
47 print("\nErste 5 Zeilen der sortierten Daten:")
48 print(df_sorted.head())
49
50 # Optional: Speichern des sortierten DataFrames in eine neue Excel-Datei
51 ausgabe_pfad = 'Fallzahlen_sortiert.xlsx'
52 try:
53     df_sorted.to_excel(ausgabe_pfad, index=False)
54     print(f"\nSortierte Daten wurden erfolgreich in '{ausgabe_pfad}'
    gespeichert.")
55 except Exception as e:
56     print(f"Ein Fehler ist aufgetreten beim Speichern der Datei: {e}")
57     '''
58
59 ### Erklärung des Skripts:
60
61 - **Importieren der Bibliothek**:
62     '''python
63     import pandas as pd
64     '''
65     'pandas' ist eine leistungsstarke Bibliothek für Datenmanipulation und -
        analyse.
66
67 - **Einlesen der Excel-Datei**:
68     '''python
69     df = pd.read_excel(datei_pfad, sheet_name=sheet_name)
70     '''
71     Diese Zeile liest das angegebene Sheet der Excel-Datei in einen DataFrame '
        df' ein.
72
73 - **Filtern der unerwünschten Zeilen**:
74     '''python
75     df_filtered = df[~df['LOR-Schlüssel'].isin(ausschluss_schluessel)]
76     '''
77     Hier werden alle Zeilen ausgeschlossen, deren 'LOR-Schlüssel' entweder
        '999900' oder '999999' ist. Das '~' Symbol negiert die Bedingung.
78
79 - **Sortieren der Daten**:
80     '''python
81     df_sorted = df_filtered.sort_values(by='Straftaten_insgesamt', ascending=
        False).reset_index(drop=True)
82     '''
83     Diese Zeile sortiert die gefilterten Daten nach der Spalte '
        Straftaten_insgesamt' in absteigender Reihenfolge. Das 'reset_index(drop=
        True)' stellt sicher, dass die Indizes neu gesetzt werden und die alten
        Indizes verworfen werden.
84
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 '
    Fallzahlen_sortiert.xlsx'. Der Parameter 'index=False' sorgt dafür, dass
    die DataFrame-Indizes nicht mitgespeichert werden.
90
91 ### Ausführen des Skripts:
92
93 Speichere das oben stehende Skript in einer Datei, z.B. 'fallzahlen_sortieren
    .py', und führe es mit Python aus:
94
95 '''bash
96 python fallzahlen_sortieren.py
97 '''
98
99 ### Beispielausgabe:
100
101 '''
102 Excel-Datei erfolgreich eingelesen.
103
104 Erste 5 Zeilen der Originaldaten:
105
106   LOR-Schlüssel      Bezirke  Straftaten_insgesamt  Raub
107   Straßenraub  Körperverletzungen_insgesamt  ...  Branddelikte_insgesamt
108   Brandstiftung  Sachbeschädigung_insgesamt  Sachbeschädigung_Graffiti
109   Rauschgiftdelikte  Kieztaten
110
111 0      10000      Mitte      83611      884
112   521      8144  ...      420
113   137      6172      976
114   3306      19212
115 1      20000  Friedrichshain-Kreuzberg      55838      923
116   606      5502  ...      298
117   85      4906      801
118   4455      12724
119 ...      ...      ...      ...
120   ...      ...      ...      ...
121   ...      ...      ...      ...
122   ...      ...      ...      ...
123
124 [14 rows x 19 columns]
125
126 Anzahl der Zeilen nach dem Ausschluss: 12
127
128 Erste 5 Zeilen der sortierten Daten:
129
130   LOR-Schlüssel      Bezirke  Straftaten_insgesamt  Raub
131   Straßenraub  Körperverletzungen_insgesamt  ...  Branddelikte_insgesamt
132   Brandstiftung  Sachbeschädigung_insgesamt  Sachbeschädigung_Graffiti
133   Rauschgiftdelikte  Kieztaten
134
135 0      10000      Mitte      83611      884
136   521      8144  ...      420
137   137      6172      976
138   3306      19212
139 1      20000  Friedrichshain-Kreuzberg      55838      923

```



```

118 ...      606      5502 ...      298
      85      4906      801
      4455      12724
      ...      ...
      ...      ...
      ...      ...
      ...      ...
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 112: API 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
      ück.

```

```

26     """
27     try:
28         df = pd.read_excel(file_path, sheet_name=sheet_name)
29         print(f"Erfolgreich das Blatt '{sheet_name}' aus '{file_path}'
gelesen.")
30         return df
31     except FileNotFoundError:
32         print(f"Fehler: Die Datei '{file_path}' wurde nicht gefunden.")
33         sys.exit(1)
34     except ValueError:
35         print(f"Fehler: Das Blatt '{sheet_name}' existiert nicht in '{
file_path}'")
36         sys.exit(1)
37     except Exception as e:
38         print(f"Ein unerwarteter Fehler ist aufgetreten: {e}")
39         sys.exit(1)
40
41 def validate_columns(df, required_columns):
42     """
43     Überprüft, ob die erforderlichen Spalten im DataFrame vorhanden sind.
44     """
45     missing_columns = [col for col in required_columns if col not in df.
columns]
46     if missing_columns:
47         print(f"Fehler: Die folgenden erforderlichen Spalten fehlen im
DataFrame: {missing_columns}")
48         sys.exit(1)
49     else:
50         print("Alle erforderlichen Spalten sind vorhanden.")
51
52 def sort_dataframe(df, sort_column, exclude_keys, key_column):
53     """
54     Sortiert den DataFrame nach 'sort_column' absteigend, wobei die Zeilen
mit 'exclude_keys'
55     am Ende verbleiben.
56     """
57     # Trennen der Zeilen, die ausgeschlossen werden sollen
58     df_excluded = df[df[key_column].isin(exclude_keys)]
59     df_to_sort = df[~df[key_column].isin(exclude_keys)]
60
61     # Sortieren der relevanten Zeilen
62     df_sorted = df_to_sort.sort_values(by=sort_column, ascending=False)
63
64     # Zusammenführen der sortierten und ausgeschlossenen Zeilen
65     df_final = pd.concat([df_sorted, df_excluded], ignore_index=True)
66
67     print("DataFrame erfolgreich sortiert.")
68     return df_final
69
70 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
114       entsprechende Fehlermeldungen aus.
115
116 3. **Funktion 'validate_columns':**
117     - Überprüft, ob die notwendigen Spalten ('Straftaten_insgesamt' und '
118       LOR-Schlüssel') im DataFrame vorhanden sind.
119     - Beendet das Skript mit einer Fehlermeldung, wenn eine erforderliche
120       Spalte fehlt.
121
122 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     ```bash
152     python sort_fallzahlen.py
153     ```
154
155 4. **Ergebnis:**
156 - Das sortierte DataFrame wird in einer neuen Excel-Datei namens '
157     Fallzahlen_sortiert.xlsx' gespeichert.
158 - Bei Fehlern erhalten Sie entsprechende Fehlermeldungen im Terminal.
159
160 ### Anpassungen
161
162 - **Andere Sheet-Namen oder Dateipfade:**
163 - Passen Sie die Variablen 'file_path' und 'sheet_name' im 'main'-Bereich
164     entsprechend an.
165
166 - **Weitere Sortierkriterien:**
167 - Wenn Sie nach zusätzlichen Kriterien sortieren möchten, können Sie die '
168     sort_values'-Funktion entsprechend erweitern.
169
170 - **Weitere Fehlersicherungen:**
171 - Sie können zusätzliche Überprüfungen hinzufügen, z.B. überprüfen, ob die

```

```
163     auszuschließenden Schlüssel tatsächlich vorhanden sind.
164 Ich hoffe, dieses Skript hilft Ihnen weiter! Falls Sie weitere Anpassungen
    oder Erklärungen benötigen, lassen Sie es mich wissen.
```

Listing 113: API 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']
37         fehlende_spalten = [spalte for spalte in benötigte_spalten if spalte
38                             not in df.columns]
```

```

36         if fehlende_spalten:
37             raise ValueError(f"Fehlende Spalten in der Excel-Datei: {
fehlende_spalten}")
38
39         # Schritt 2: Trennen der Bezirke und Nicht-Bezirke
40         nicht_bezirke_keys = [999900, 999999]
41
42         # Sicherstellen, dass 'LOR-Schlüssel' als numerisch betrachtet wird
43         df['LOR-Schlüssel'] = pd.to_numeric(df['LOR-Schlüssel'], errors='
coerce')
44
45         bezirke_df = df[~df['LOR-Schlüssel'].isin(nicht_bezirke_keys)].copy()
46         nicht_bezirke_df = df[df['LOR-Schlüssel'].isin(nicht_bezirke_keys)].
copy()
47
48         # Schritt 3: Sortieren der Bezirke nach 'Straftaten_insgesamt'
absteigend
49         bezirke_df.sort_values(by='Straftaten_insgesamt', ascending=False,
inplace=True)
50
51         # Schritt 4: Zusammenfügen der sortierten Bezirke mit den Nicht-
Bezirken
52         sortiertes_df = pd.concat([bezirke_df, nicht_bezirke_df],
ignore_index=True)
53
54         print("Daten erfolgreich sortiert.")
55         return sortiertes_df
56
57     except FileNotFoundError:
58         print(f"Datei '{file_path}' wurde nicht gefunden.")
59     except ValueError as ve:
60         print(f"Wertfehler: {ve}")
61     except Exception as e:
62         print(f"Ein unerwarteter Fehler ist aufgetreten: {e}")
63
64 # Anwendung des Skripts
65 if __name__ == "__main__":
66     datei_pfad = 'Fallzahlen.xlsx'
67     sheet = 'Fallzahlen_2023'
68
69     sortiertes_dataframe = sort_fallzahlen_excel(datei_pfad, sheet)
70
71     if sortiertes_dataframe is not None:
72         # Schritt 5: Optional - Speichern des sortierten DataFrames in einer
neuen Excel-Datei
73         ausgabe_pfad = 'Sortierte_Fallzahlen.xlsx'
74         try:
75             sortiertes_dataframe.to_excel(ausgabe_pfad, index=False)
76             print(f"Sortiertes DataFrame wurde erfolgreich in '{ausgabe_pfad
}' 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 '''
128
129 Stellen Sie sicher, dass sich die Excel-Datei 'Fallzahlen.xlsx' im gleichen
130 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 114: API 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)]
43
44     # 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
      Ausschlussbedingungen korrekt anwenden zu können.
90     - Trennt die DataFrame in zwei Teile:
91         - 'df_bezirke': alle Zeilen außer denen mit 'LOR-Schlüssel' 999900 und
      999999.
92         - 'df_excluded': die auszuschließenden Zeilen.
93
94 5. **Sortieren der Bezirke:**
95     - Sortiert 'df_bezirke' nach 'Straftaten_insgesamt' in absteigender
      Reihenfolge.
96     - Fehlerbehandlung für mögliche Probleme beim Sortieren.
97
98 6. **Zusammenführen der DataFrames:**
99     - Kombiniert das sortierte Bezirke-DataFrame mit den ausgeschlossenen
      Zeilen am Ende.
100
101 7. **Optional: Ausgabe oder Speicherung:**
102     - Zeigt die ersten fünf Zeilen des finalen DataFrames an.
103     - Speichert das sortierte DataFrame in eine neue Excel-Datei '
      Fallzahlen_sortiert.xlsx'.
104     - Fehlerbehandlung beim Speichern der Datei.
105
106 ### Hinweise
107
108 - **Robustheit:** Das Skript enthält umfassende Fehlerbehandlungen, die
      sicherstellen, dass fehlende Dateien, fehlende Spalten oder Datenprobleme
      ordnungsgemäß gehandhabt werden.
109
110 - **Anpassung:** Sie können den 'output_file' Namen ändern oder weitere
      Verarbeitungsschritte hinzufügen, je nach Bedarf.
111
112 - **Ausführung:** Stellen Sie sicher, dass Sie die notwendigen Python-Pakete
      installiert haben, insbesondere 'pandas' und 'openpyxl' (für das Lesen und
      Schreiben von Excel-Dateien). Sie können fehlende Pakete mit 'pip install
      pandas openpyxl' installieren.
113
114 '''bash
115 pip install pandas openpyxl
116 '''
117
118 ### Beispiel Ausführung
119
120 Nachdem Sie das Skript gespeichert haben (z.B. als 'sort_fallzahlen.py'), kö
      nnen Sie es über die Kommandozeile ausführen:
121
122 '''bash
123 python sort_fallzahlen.py
124 '''
125
126 Wenn alles korrekt eingerichtet ist, wird das sortierte DataFrame erstellt
      und in der Datei 'Fallzahlen_sortiert.xlsx' gespeichert.

```

Listing 115: API 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)]

```

```

43     if excluded_df.empty:
44         print("Warning: No rows found with LÖR-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',
50         ascending=False)
51         print("Successfully sorted the included data by 'Straftaten_insgesamt'
52         in descending order.")
53     except KeyError:
54         print("Error: The column 'Straftaten_insgesamt' does not exist in the
55         data.")
56         return pd.DataFrame()
57     except Exception as e:
58         print(f"An unexpected error occurred during sorting: {e}")
59         return pd.DataFrame()
60
61     # Concatenate the sorted included data with the excluded data
62     sorted_df = pd.concat([included_sorted_df, excluded_df], ignore_index=
63     True)
64     print("Successfully concatenated the excluded rows at the end of the
65     sorted data.")
66
67     return sorted_df
68
69 if __name__ == "__main__":
70     # Define the Excel file path and sheet name
71     excel_file = 'Fallzahlen.xlsx'
72     sheet = 'Fallzahlen_2023'
73
74     # Get the sorted DataFrame
75     sorted_fallzahlen_df = sort_fallzahlen_excel(excel_file, sheet)
76
77     # Check if the DataFrame is not empty before proceeding
78     if not sorted_fallzahlen_df.empty:
79         print("Sorted DataFrame:")
80         print(sorted_fallzahlen_df)
81         # Optionally, you can save the sorted DataFrame to a new Excel file
82         # or another format
83         # sorted_fallzahlen_df.to_excel('Sorted_Fallzahlen.xlsx', index=False
84         # )
85     else:
86         print("No data to display due to previous errors.")
87
88     """
89
90     ### Erklärung des Skripts
91
92     1. **Importieren der notwendigen Bibliotheken:**
93     - 'pandas' wird benötigt, um die Excel-Datei zu lesen und mit DataFrames
94     zu arbeiten.
95

```

```

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

```

```

126     Verzeichnis wie das Skript liegt oder geben Sie den vollständigen Pfad zur
127     Datei an.
128
129 2. Führen Sie das Skript mit Python aus:
130     '''bash
131     python sort_fallzahlen.py
132     '''
133     (Angenommen, Sie speichern das Skript als 'sort_fallzahlen.py')
134
135 3. Das Skript gibt das sortierte DataFrame in der Konsole aus. Optional kö-
136     nnen Sie das Ergebnis in eine neue Datei speichern, indem Sie die
137     entsprechende Zeile entkommentieren.

```

Listing 116: API Response Testfall 1 Prompt 3 Ausführung 4

```

1 Hier ist ein robustes Python-Skript, das die von dir beschriebenen Schritte
2   ausführt. Es verwendet die 'pandas'-Bibliothek zum Einlesen und
3   Verarbeiten der Excel-Datei. Das Skript beinhaltet umfassende
4   Fehlerbehandlungen, um sicherzustellen, dass fehlende Spalten oder andere
5   potenzielle Probleme angemessen behandelt werden.
6
7   '''python
8   import pandas as pd
9
10  def sort_fallzahlen_excel(file_path: str, sheet_name: str) -> pd.DataFrame:
11      """
12      Liest die Daten aus dem angegebenen Excel-Sheet, sortiert sie nach '
13      Straftaten_insgesamt' absteigend,
14      wobei die Zeilen mit den LOR-Schlüsseln 999900 und 999999 am Ende des
15      DataFrames stehen.
16
17      Parameters:
18      - file_path: Pfad zur Excel-Datei.
19      - sheet_name: Name des Sheets, das eingelesen werden soll.
20
21      Returns:
22      - Ein sortierter Pandas DataFrame.
23      """
24      try:
25          # Schritt 1: Einlesen der Excel-Datei
26          df = pd.read_excel(file_path, sheet_name=sheet_name)
27          print(f"Erfolgreich das Sheet '{sheet_name}' aus '{file_path}'
28          eingelesen.")
29      except FileNotFoundError:
30          raise FileNotFoundError(f"Die Datei '{file_path}' wurde nicht
31          gefunden.")
32      except ValueError:
33          raise ValueError(f"Das Sheet '{sheet_name}' existiert nicht in '{
34          file_path}'")
35      except Exception as e:
36          raise Exception(f"Ein unerwarteter Fehler ist beim Einlesen der Excel
37          -Datei aufgetreten: {e}")
38

```

```

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
80     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.

```



```

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     '''

```

```

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 117: API Response Testfall 1 Prompt 3 Ausführung 5

11.3.9 Komplette Antworten der API Requests Testfall 2

```

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

```

```

79
80 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 118: API 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
41 # Kombinieren aller relevanten Daten
42 if daten_liste:
43     alle_daten = pd.concat(daten_liste)
44 else:
45     print("Keine relevanten Daten gefunden.")
46     exit(1)
47
48 # Aggregation der Straftaten pro Bezirk
49 aggregierte_daten = alle_daten.groupby('Bezirke', as_index=False)['
50     Straftaten_insgesamt'].sum()
51
52 # Sortieren der Bezirke nach der Anzahl der Straftaten (absteigend)
53 aggregierte_daten_sortiert = aggregierte_daten.sort_values(by='
54     Straftaten_insgesamt', ascending=False).reset_index(drop=True)
55
56 # Optional: Anzeige des resultierenden DataFrames
57 print(aggregierte_daten_sortiert)
58
59 # Optional: Speichern des DataFrames in eine neue Excel-Datei
60 aggregierte_daten_sortiert.to_excel('Aggregierte_Fallzahlen.xlsx', index=
61     False)
62
63 """
64
65 ### Erklärung des Skripts:
66
67 1. **Importieren der Bibliotheken:**
68     - 'pandas' wird zum Verarbeiten der Excel-Datei verwendet.
69

```

```

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 119: API 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)['
78             Straftaten_insgesamt'].sum()
79         '''
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',
84             ascending=False)
85         sortiert_df = sortiert_df.reset_index(drop=True)
86         '''
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
100   gespeichert werden. Die entsprechenden Zeilen können entkommentiert werden
101   , um diese Funktionalität zu aktivieren.
102
103 ### Hinweise:
104
105 - **Duplikate:** Falls ein Bezirk in mehreren Sheets vorkommt und du die
106   Einträge nicht aggregieren möchtest, kannst du die Zeile zur Entfernung
107   von Duplikaten ('drop_duplicates') nutzen. Andernfalls aggregierst du die
108   Straftaten pro Bezirk unabhängig von den einzelnen Sheets.
109
110 - **Weitere Analysen:** Du kannst den 'aggregiert_df' oder 'sortiert_df'
111   weitere Analysen unterziehen oder zusätzliche Spalten berechnen, je nach
112   Bedarf.
113
114 - **Fehlerbehandlung:** Für größere Projekte oder zur Produktion könnten zusätz-
115   liche Schritte zur Fehlerbehandlung sinnvoll sein, z.B. Überprüfen, ob
116   bestimmte Spalten existieren oder mit fehlenden Daten umgehen.
117
118 Dieses Skript sollte dir einen guten Startpunkt bieten, um die Daten aus
119 deiner Excel-Datei zu verarbeiten und die gewünschten Informationen zu
120 extrahieren.

```

Listing 120: API Response Testfall 2 Prompt 1 Ausführung 3

```

1  Gerne! Hier ist ein Python-Skript, das alle Sheets aus der Excel-Datei '
2    Fallzahlen.xlsx' liest, die Daten nach der Anzahl der Straftaten pro
3    Bezirk aggregiert, sortiert und in einem Pandas DataFrame speichert.
4
5  '''python
6  import pandas as pd
7
8  # Pfad zur Excel-Datei
9  excel_file = 'Fallzahlen.xlsx'
10
11 # Schritt 1: Alle Sheets der Excel-Datei einlesen
12 # Das Argument sheet_name=None liest alle Sheets und gibt ein Dictionary zurück
13 all_sheets = pd.read_excel(excel_file, sheet_name=None)
14
15 # Schritt 2: Daten aus allen Sheets zusammenführen
16 # Wir gehen davon aus, dass alle Sheets die gleiche Struktur haben
17 # Daher können wir sie einfach mit pd.concat zusammenfügen
18 combined_df = pd.concat(all_sheets.values(), ignore_index=True)
19
20 # Schritt 3: Daten nach 'Bezirke' gruppieren und 'Straftaten_insgesamt'
21     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 121: API 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.

```
'''bash
pip install pandas openpyxl
'''
```

2. ****Erstelle das Python-Skript****:

```
'''python
import pandas as pd

def lese_und_sortiere_fallzahlen(excel_datei):
    """
    Liest alle Sheets aus der gegebenen Excel-Datei, kombiniert die Daten
    ,
    sortiert sie nach der Gesamtanzahl der Straftaten pro Bezirk und
    gibt das sortierte DataFrame zurück.

    :param excel_datei: Pfad zur Excel-Datei 'Fallzahlen.xlsx'
    :return: Sortiertes pandas DataFrame
    """
    try:
        # Alle Sheets lesen; sheet_name=None gibt ein Dictionary zurück
        alle_sheets = pd.read_excel(excel_datei, sheet_name=None)
        print(f"Anzahl der gelesenen Sheets: {len(alle_sheets)}")
    except FileNotFoundError:
        print(f"Die Datei '{excel_datei}' wurde nicht gefunden.")
        return None
    except Exception as e:
        print(f"Ein Fehler ist aufgetreten: {e}")
        return None

    # Kombinieren aller DataFrames in einen
    try:
        kombiniertes_df = pd.concat(alle_sheets.values(), ignore_index=
True)

        print(f"Anzahl der Zeilen nach dem Kombinieren: {len(
kombiniertes_df)}")
    except ValueError as ve:
        print(f"Fehler beim Kombinieren der Daten: {ve}")
        return None

    # Sicherstellen, dass die Spalte 'Straftaten_insgesamt' numerisch ist
    if not pd.api.types.is_numeric_dtype(kombiniertes_df['
Straftaten_insgesamt']):
        kombiniertes_df['Straftaten_insgesamt'] = pd.to_numeric(
            kombiniertes_df['Straftaten_insgesamt'], errors='coerce'
        )
        kombiniertes_df = kombiniertes_df.dropna(subset=['
Straftaten_insgesamt'])
```

```

51     # Sortieren nach 'Straftaten_insgesamt' absteigend
52     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
97   Verzeichnis wie das Skript befindet oder passe den Pfad entsprechend an.
98 3. **Führe das Skript aus**:
99
100     '''bash
101     python sortiere_fallzahlen.py
102     '''
103
104 4. **Optional**: Wenn du das sortierte Ergebnis auch als CSV-Datei speichern
105   möchtest, entferne das Kommentarzeichen ('#') vor den entsprechenden
106   Zeilen im Skript.
107
108 ### Beispielausgabe:
109
110 Die Ausgabe wird ein 'DataFrame' sein, das nach der Gesamtzahl der Straftaten
111 pro Bezirk sortiert ist, beginnend mit dem Bezirk mit den meisten
112 Straftaten. Hier ist eine vereinfachte Darstellung basierend auf deinem
113 Beispiel:
114
115 '''
116
117     LOR-Schlüssel          Bezirke  Straftaten_insgesamt  Raub
118     Straßenraub ...  Rauschgiftdelikte  Kieztaten
119 0          999999          Berlin (PKS gesamt)          536695  5389
120          3333 ...          17502          119737
121 1          10000          Mitte          83611  884
122          521 ...          3306          19212
123 2          80000          Neukölln          45313  508
124          294 ...          1869          10253
125 ...
126 '''
127
128 Dieses Skript sollte dir eine solide Grundlage bieten, um deine Daten
129 effizient zu verarbeiten und nach den gewünschten Kriterien zu sortieren.
130 Wenn du weitere Anpassungen benötigst, lass es mich wissen!

```

Listing 122: API Response Testfall 2 Prompt 1 Ausführung 5

```

1 Natürlich, ich helfe dir gerne dabei, ein Python-Skript zu erstellen, das
2   deine Anforderungen erfüllt. Das Skript wird die Daten aus allen Sheets
3   der Excel-Datei 'Fallzahlen.xlsx' zusammenführen, die unerwünschten 'LOR-
4   Schlüssel'-Zeilen entfernen, die Daten nach der Spalte '
5   Straftaten_insgesamt' sortieren und das Ergebnis in einem Pandas DataFrame
6   speichern.
7
8 Hier ist ein vollständiges Python-Skript, das diese Schritte ausführt:
9
10 '''python
11 import pandas as pd
12
13 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())

```

```

58     # Optional: Speichern Sie den finalen DataFrame als neue Excel-Datei
59     # 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
       von DataFrames.
72     - Anschließend werden alle DataFrames zu einem einzigen DataFrame
       kombiniert.
73
74 3. **Entfernen der unerwünschten 'LOR-Schlüssel':**
75     '''python
76     filtered_df = kombiniert_df[~kombiniert_df['LOR-Schlüssel'].isin([999900,
77     999999])]
78     '''
79     - Diese Zeile filtert alle Zeilen heraus, deren 'LOR-Schlüssel' entweder
       '999900' oder '999999' ist.
80
81 4. **Sortieren des DataFrames:**
82     '''python
83     sorted_df = filtered_df.sort_values(by='Straftaten_insgesamt', ascending=
84     False).reset_index(drop=True)
85     '''
86     - Sortiert den DataFrame basierend auf der Spalte 'Straftaten_insgesamt'
       in absteigender Reihenfolge.
87     - 'reset_index(drop=True)' setzt den Index zurück, sodass er fortlaufend
       von 0 beginnt.
88
89 5. **Hauptblock des Skripts:**
90     - Definiert den Pfad zur Excel-Datei.
91     - Ruft die oben definierte Funktion auf und speichert das Ergebnis in '
       final_df'.
92     - Optional zeigt es die ersten paar Zeilen des finalen DataFrames an.
93     - Optional kann der finale DataFrame in eine neue Excel-Datei gespeichert
       werden (die entsprechende Zeile ist auskommentiert).
94
95 ### Hinweise:
96
97 - **Abhängigkeiten installieren:**
98     Stelle sicher, dass du die benötigten Bibliotheken installiert hast. Du
99     kannst 'pandas' mit folgendem Befehl installieren:
100     '''bash
101     pip install pandas
102     '''

```



```

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

```

```

71
72     # Optional: Speichere den DataFrame in eine neue Excel- oder CSV-
       Datei
73     # final_df.to_excel('Zusammengefuegte_Fallzahlen.xlsx', index=False)
74     # final_df.to_csv('Zusammengefuegte_Fallzahlen.csv', index=False)
75 '''
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

```

```

105         angezeigt werden:
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 124: API 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

```

```

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 # 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

```

```

56 # Sortieren nach 'Straftaten_insgesamt' absteigend
57 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 (999900 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 125: API 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-
    Schlüssel,
19     fasst die Daten der Bezirke zusammen, sortiert sie nach
    Straftaten_insgesamt
20     und gibt das resultierende DataFrame zurück.

```

```

21
22 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
55         erforderliche Spalten. Überspringe dieses Sheet.")
56         continue
57
58     # Filtere die unerwünschten LOR-Schlüssel
59     df_filtered = df[~df['LOR-Schlüssel'].isin(exclude_lor_keys)].copy()
60     print(f"Anzahl der Bezirke nach Filtern in Sheet '{sheet_name}': {len
61     (df_filtered)}")
62
63     # Optional: Falls du weitere Daten aggregieren möchtest, kannst du
64     hier weitere Schritte hinzufügen
65
66     filtered_dfs.append(df_filtered)
67
68 if not filtered_dfs:
69     print("Keine gültigen Daten zum Zusammenfügen gefunden.")
70     return None
71
72 # Füge alle gefilterten DataFrames zusammen

```



```

69     combined_df = pd.concat(filtered_dfs, ignore_index=True)
70     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      LOR-Schlüssel      Bezirke  Straftaten_insgesamt
142 0          10000          Mitte          83611
143 1          20000  Friedrichshain-Kreuzberg          55838
144 2          80000          Neukölln          45313
145 3          30000          Pankow          39164
146 4          40000  Charlottenburg-Wilmersdorf          53630
147 ...
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
      Spalten aggregieren möchtest, kannst du die 'groupby'-Funktion
      entsprechend erweitern.
155
156 - **Fehlerbehandlung:** Das Skript enthält grundlegende Fehlerbehandlungen,
      um Probleme beim Laden der Datei oder fehlende Spalten zu erkennen.
157
158 - **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.
159
160 Ich hoffe, dieses Skript hilft dir weiter! Bei weiteren Fragen stehe ich
      gerne zur Verfügung.

```

Listing 126: API 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
      ngig von der Excel-Version).
8
9 2. **Lese alle Sheets der Excel-Datei ein**:
10    - Nutze 'pd.ExcelFile', um die Sheet-Namen zu erhalten.
11    - Iteriere über alle Sheets, lese sie ein und filtere die unerwünschten
      Zeilen.
12
13 3. **Daten bereinigen und zusammenführen**:
14    - Entferne Zeilen mit den 'LOR-Schlüsseln' 999900 und 999999.
15    - Wähle relevante Spalten aus.
16    - Füge die Daten aus allen Sheets zusammen.
17
18 4. **Sortiere die Daten**:
19    - Gruppieren nach 'LOR-Schlüssel' und 'Bezirke'.
20    - Summiere die 'Straftaten_insgesamt' für jeden Bezirk.
21    - Sortiere die Bezirke basierend auf der summierten 'Straftaten_insgesamt
      '-Spalte.
22
23 5. **Speichere das Ergebnis in einem DataFrame**.
24
25 ### Python-Skript:
26
27 '''python
28 import pandas as pd

```

```

29
30 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:

```

```

76         print(f"Fehler beim Verarbeiten des Sheets '{sheet_name}': {e}")
77         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     # Gruppiere nach 'LOR-Schlüssel' und 'Bezirke' und summiere '
Straftaten_insgesamt'
87     grouped_df = combined_df.groupby(['LOR-Schlüssel', 'Bezirke'], as_index=
False)['Straftaten_insgesamt'].sum()
88
89     # Sortiere nach 'Straftaten_insgesamt' absteigend
90     sorted_df = grouped_df.sort_values(by='Straftaten_insgesamt', ascending=
False).reset_index(drop=True)
91
92     return sorted_df
93
94 if __name__ == "__main__":
95     # Pfad zur Excel-Datei
96     excel_file_path = 'Fallzahlen.xlsx'
97
98     # Verarbeite die Excel-Datei und erhalte den finalen DataFrame
99     final_df = process_fallzahlen_excel(excel_file_path)
100
101     if final_df is not None:
102         # Zeige die ersten paar Zeilen des finalen DataFrames an
103         print(final_df)
104
105         # Optional: Speichere den finalen DataFrame in einer neuen Excel-
oder CSV-Datei
106         # final_df.to_excel('Zusammengefasste_Fallzahlen.xlsx', index=False)
107         # final_df.to_csv('Zusammengefasste_Fallzahlen.csv', index=False)
108     '''
109
110     ### Erklärung des Skripts:
111
112     1. **Funktion 'process_fallzahlen_excel':**
113         - **Parameter**: 'file_path' - der Pfad zur Excel-Datei.
114         - **Rückgabe**: Ein 'DataFrame', das die zusammengeführten und sortierten
Daten enthält.
115         - **Fehlerbehandlung**: Das Skript überprüft, ob die Datei existiert und
ob die erforderlichen Spalten in jedem Sheet vorhanden sind.
116
117     2. **Hauptteil des Skripts ('if __name__ == "__main__":')**:
118         - Definiert den Pfad zur Excel-Datei.
119         - Ruft die Verarbeitungsfunktion auf.
120         - Gibt den finalen 'DataFrame' aus.
121         - Optional: Speichert das Ergebnis in einer neuen Excel- oder CSV-Datei.

```

```

122
123 ### 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 127: API 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     # Setzen von 'LOR-Schlüssel' und 'Bezirke' als Index
30     df = df.set_index(['LOR-Schlüssel', 'Bezirke'])
31     df_liste.append(df)
32
33 if not df_liste:
34     print("Keine gültigen Sheets zum Verarbeiten gefunden.")
35     return
36
37 # Alle DataFrames zusammenfügen und numerische Spalten akkumulieren
38 zusammengefuegt_df = pd.concat(df_liste)
39 # Gruppieren nach den Index-Spalten und summieren der numerischen Werte
40 akkumuliert_df = zusammengefuegt_df.groupby(level=['LOR-Schlüssel', '
41 Bezirke']).sum().reset_index()
42
43 # 3. DataFrame sortieren nach 'Straftaten_insgesamt', spezielle Zeilen am
44 Ende platzieren
45 # Definieren der speziellen LOR-Schlüssel
46 spezielle_keys = [999900, 999999]
47
48 # Trennen der regulären und speziellen Zeilen
49 regulär_df = akkumuliert_df[~akkumuliert_df['LOR-Schlüssel'].isin(
50 spezielle_keys)]
51 spezielle_df = akkumuliert_df[akkumuliert_df['LOR-Schlüssel'].isin(
52 spezielle_keys)]
53
54 # Sortieren der regulären Zeilen nach 'Straftaten_insgesamt' absteigend
55 sortiert_regulär_df = regulär_df.sort_values(by='Straftaten_insgesamt',
56 ascending=False)
57
58 # Kombinieren der sortierten regulären Zeilen mit den speziellen Zeilen
59 endgültiges_df = pd.concat([sortiert_regulär_df, spezielle_df],
60 ignore_index=True)
61
62 # 4. Sortiertes DataFrame zurückgeben (hier als Ausgabe angezeigt)
63 print(endgültiges_df)
64
65 if __name__ == '__main__':
66     main()
67
68 """
69
70 ### Erklärung der Schritte:
71
72 1. **Excel-Datei einlesen**:

```

```

66 - Das Skript liest alle Sheets der Excel-Datei 'Fallzahlen.xlsx' ein und
    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 128: API 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:

```

```

49     print(f"Fehler beim Zusammenführen der DataFrames: {e}")
50     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 = [9999900, 9999999]
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

```

```

92     Excel-Datei ein und speichert sie in einem Dictionary, wobei die Sheet-
93     Namen die Schlüssel sind.
94
95 2. **DataFrames zusammenführen und Straftaten akkumulieren:**
96     - Alle einzelnen DataFrames werden mit 'pd.concat' zu einem großen
97     DataFrame zusammengefügt.
98     - Anschließend wird das DataFrame nach den Spalten ''LOR-Schlüssel'' und ''Bezirke''
99     gruppiert, und alle numerischen Spalten werden summiert, um die
100     akkumulierten Werte der Straftaten zu erhalten.
101
102 3. **Sortieren nach ''Straftaten_insgesamt'', spezielle LOR-Schlüssel am Ende
103     :**
104     - Die Zeilen mit den LOR-Schlüsseln '999900' und '999999' werden separat
105     gehalten.
106     - Die restlichen Zeilen werden nach der Spalte ''Straftaten_insgesamt'' in
107     absteigender Reihenfolge sortiert.
108     - Schließlich werden die speziellen Zeilen ans Ende des DataFrames angehä
109     ngt.
110
111 4. **Rückgabe des sortierten DataFrames:**
112     - Das sortierte DataFrame wird zurückgegeben und kann weiter verwendet
113     oder angezeigt werden.
114     - Optional können Sie das Ergebnis auch in eine neue Excel-Datei speichern
115     , indem Sie die auskommentierte Zeile 'to_excel' verwenden.
116
117 ### Hinweise:
118
119 - **Fehlerbehandlung:** Das Skript enthält grundlegende Fehlerbehandlungen,
120   um häufige Probleme wie fehlende Dateien oder Spalten abzufangen.
121 - **Anpassungen:** Je nach Struktur Ihrer Excel-Datei und den spezifischen
122   Anforderungen könnten weitere Anpassungen erforderlich sein.
123 - **Speichern des Ergebnisses:** Falls Sie das Ergebnis in eine neue Datei
124   speichern möchten, entfernen Sie den Kommentar vor der entsprechenden
125   Zeile.
126
127 Ich hoffe, dieses Skript erfüllt Ihre Anforderungen. Bei weiteren Fragen oder
128 Anpassungswünschen stehe ich gerne zur Verfügung!

```

Listing 129: API 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

```

```

14     Spalten akkumulieren
    gruppiert_df = concatenated_df.groupby(['LOR-Schlüssel', 'Bezirke'],
15     as_index=False).sum()
16
17     # 3. DataFrame sortieren nach 'Straftaten_insgesamt', dabei LOR-Schlüssel
    999900 und 999999 am Ende lassen
18     # Definieren der auszuschließenden LOR-Schlüssel
    ausschluss_keys = [999900, 999999]
19
20     # DataFrame ohne die auszuschließenden LOR-Schlüssel
21     zu_sortieren_df = gruppiert_df[~gruppiert_df['LOR-Schlüssel'].isin(
    ausschluss_keys)]
22
23     # DataFrame mit den auszuschließenden LOR-Schlüsseln
24     ausgeschlossen_df = gruppiert_df[gruppiert_df['LOR-Schlüssel'].isin(
    ausschluss_keys)]
25
26     # Sortieren nach 'Straftaten_insgesamt' in absteigender Reihenfolge
27     sortiert_df = zu_sortieren_df.sort_values(by='Straftaten_insgesamt',
    ascending=False)
28
29     # Ausschließenere Zeilen am Ende des sortierten DataFrames anhängen
30     final_df = pd.concat([sortiert_df, ausgeschlossen_df], ignore_index=True)
31
32     # 4. Sortiertes DataFrame zurückgeben (hier wird es angezeigt)
33     print(final_df)
34
35 if __name__ == '__main__':
36     main()
37
38
39 ### Erklärung der Schritte:
40
41 1. **Excel-Datei einlesen:**
42     - Mit 'pd.read_excel' wird die Excel-Datei 'Fallzahlen.xlsx' eingelesen.
43     - 'sheet_name=None' liest alle Sheets und speichert sie in einem
    Dictionary, wobei die Sheet-Namen die Schlüssel sind.
44
45 2. **DataFrames zusammenführen:**
46     - Die einzelnen DataFrames werden mit 'pd.concat' vertikal zusammengeführt
    .
47     - Anschließend wird das kombinierte DataFrame nach den Spalten 'LOR-Schlüssel' und 'Bezirke' gruppiert.
48     - Mit '.sum()' werden die numerischen Spalten akkumuliert, sodass pro
    Bezirk die Gesamtwerte der Straftaten stehen.
49
50 3. **Sortierung:**
51     - Definiert die LOR-Schlüssel, die nicht sortiert werden sollen ('999900'
    und '999999').
52     - Trennt das DataFrame in zwei Teile: Einen Teil, der sortiert werden soll
    , und einen Teil, der am Ende bleiben soll.
53     - Sortiert den ersten Teil nach 'Straftaten_insgesamt' in absteigender

```

```

    Reihenfolge.
54 - 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 130: API 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)

```

```

61
62     # 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 '

```

```

Fallzahlen_sortiert.xlsx'.
99
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 131: API 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

```



```

26
27 # 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
    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.")
34         continue
35     df_list.append(df)
36
37 if not df_list:
38     print("Keine gültigen Sheets gefunden.")
39     return
40
41 # Schritt 2: DataFrames zusammenführen (joinen) auf 'LOR-Schlüssel' und '
    Bezirke'
42 merged_df = df_list[0]
43 for df in df_list[1:]:
44     merged_df = pd.merge(merged_df, df, on=['LOR-Schlüssel', 'Bezirke'],
    how='outer', suffixes=('', '_dup'))
45
46     # Entfernen von doppelten Spalten, falls vorhanden
47     dup_columns = [col for col in merged_df.columns if col.endswith('_dup
    ')]
48     if dup_columns:
49         merged_df.drop(columns=dup_columns, inplace=True)
50
51 # Schritt 3: Sortieren nach 'Straftaten_insgesamt', wobei 999900 und
    999999 am Ende stehen
52 # Zuerst sicherstellen, dass 'Straftaten_insgesamt' numerisch ist
53 merged_df['Straftaten_insgesamt'] = pd.to_numeric(merged_df['
    Straftaten_insgesamt'], errors='coerce')
54
55 # Trennen der DataFrames
56 districts_df = merged_df[~merged_df['LOR-Schlüssel'].isin([999900,
    999999])]
57 non_districts_df = merged_df[merged_df['LOR-Schlüssel'].isin([999900,
    999999])]
58
59 # Sortieren der Bezirke
60 sorted_districts_df = districts_df.sort_values(by='Straftaten_insgesamt',
    ascending=False)
61
62 # Kombinieren der sortierten Bezirke mit den nicht zuordenbaren Einträgen
63 final_df = pd.concat([sorted_districts_df, non_districts_df],
    ignore_index=True)
64
65 return final_df
66

```

```

67 if __name__ == "__main__":
68     # 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.

```

105
106 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 132: API Response Testfall 2 Prompt 3 Ausführung 5

11.3.10 Komplette Antworten der API Requests Testfall 3

```
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     # Filtere nur die Bezirke, die zu Berlin gehören und nicht die
    Gesamtzeile oder nicht zuzuordnend
39     # Annahme: 'Stadtgebiet Berlin, nicht zuzuordnen' und 'Berlin (PKS gesamt
    )' sollen ausgeschlossen werden
40     relevante_bezirke = df[~df['Bezirke'].isin(['Berlin (PKS gesamt)', '
    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 133: API 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 '''

```

```

50
51 ### Erklärung des Skripts:
52
53 1. **Importieren der Bibliotheken:**
54 - 'pandas' wird verwendet, um die Excel-Datei zu lesen und zu manipulieren
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
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94

```

	Bezirke	Straftaten_insgesamt	Prozentual_Anteil (%)
0	Mitte	83611	15.58
1	Friedrichshain-Kreuzberg	55838	12.39
2	Pankow	39164	7.31
3	Charlottenburg-Wilmersdorf	53630	10.00
4	Spandau	25564	4.77
5	Steglitz-Zehlendorf	23719	4.42
6	Tempelhof-Schöneberg	41776	7.79
7	Neukölln	45313	8.46
8	Treptow-Köpenick	26099	4.87
9	Marzahn-Hellersdorf	26649	4.98
10	Lichtenberg	31563	5.89
11	Reinickendorf	27344	5.10
12	Stadtgebiet Berlin, nicht zuzuordnen	56425	10.53

```

89 Die Ergebnisse wurden in 'Straftaten_Prozentual_Anteil.xlsx' gespeichert.
90 '''
91
92 ### Hinweise:
93
94 - **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.
95
96 - **Fehlerbehandlung:** Das Skript überprüft, ob die Gesamtzeile vorhanden
    ist. Sie können zusätzliche Fehlerprüfungen hinzufügen, je nach Bedarf.
97
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 134: API 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 Gesamtstraftaten:**
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
94         Bezirke  Straftaten_insgesamt  Prozentualer_Anteil (%)
95 0             Mitte                83611                xx.xx
96 1  Friedrichshain-Kreuzberg        55838                xx.
97    xx
98 2             Pankow                39164                xx.xx
99 ...             ...                ...                ...
100 '''
101
102 Jeder Bezirk wird zusammen mit der Gesamtanzahl der Straftaten und dem
      berechneten prozentualen Anteil angezeigt.
103
104 ### Speicherung der Ergebnisse
105
106 Falls Sie die Ergebnisse in eine neue Excel-Datei speichern möchten,
      entkommentieren Sie die entsprechende Zeile am Ende des Skripts:
107
108 '''python
109 ergebnis.to_excel('Prozentuale_Anteile_Straftaten.xlsx', index=False)
110 '''
111
112 Dies erstellt eine Datei namens 'Prozentuale_Anteile_Straftaten.xlsx' im
      selben Verzeichnis wie das Skript.
113
114 ---
115
116 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 135: API 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'] /
47         total_crimes) * 100
48
49     # Optional: Auf zwei Dezimalstellen runden
50     bezirke_df['Prozentualer_Anteil'] = bezirke_df['Prozentualer_Anteil'].
51     round(2)
52
53     # Ergebnis anzeigen
54     print("Prozentualer Anteil der Straftaten nach Bezirk:")
55     print(bezirke_df[['Bezirke', 'Straftaten_insgesamt', 'Prozentualer_Anteil']])
56
57     # Optional: Ergebnis in eine neue Excel-Datei speichern
58     output_datei = 'Prozentuale_Anteile_Straftaten.xlsx'
59     try:
60         bezirke_df[['Bezirke', 'Straftaten_insgesamt', 'Prozentualer_Anteil']
61         ].to_excel(output_datei, index=False)
62         print(f"\nDie Ergebnisse wurden in '{output_datei}' gespeichert.")
63     except Exception as e:
64         print(f"Ein Fehler ist beim Speichern der Ergebnisdatei aufgetreten:
65         {e}")
66
67 if __name__ == "__main__":
68     # Pfad zur Excel-Datei
69     dateipfad = 'Fallzahlen.xlsx'
70
71     # Prozentuale Anteile berechnen
72     berechne_prozentuale_anteile(dateipfad)
73
74 """
75
76 ### Erklärung des Skripts
77
78 1. **Importieren der Bibliothek:**
79     - 'pandas' wird verwendet, um die Excel-Datei zu lesen und die Daten zu
80     verarbeiten.
81
82 2. **Funktion 'berechne_prozentuale_anteile':**
83     - **Parameter:**
84         - 'dateipfad': Der Pfad zur Excel-Datei 'Fallzahlen.xlsx'.
85         - 'sheet_name': Der Name des Sheets, standardmäßig auf 'Fallzahlen_2023'
86         gesetzt.
87
88     - **Schritte:**
89         1. **Lesen der Excel-Datei:**
90             - Versucht die Excel-Datei zu lesen. Falls die Datei nicht gefunden
91             wird oder ein anderer Fehler auftritt, wird eine Fehlermeldung ausgegeben.
92
93         2. **Bestimmen der Gesamtanzahl der Straftaten:**
94             - Sucht nach der Zeile, die den Gesamtwert für Berlin enthält ('Bezirke' == 'Berlin (PKS gesamt)').
95             - Falls diese Zeile nicht gefunden wird, summiert das Skript die
96             Straftaten aller Bezirke, außer denen, die ausgeschlossen werden sollen.

```

```

88
89     3. **Filtern der Bezirke:**
90         - Entfernt Zeilen, die nicht einem spezifischen Bezirk zugeordnet
          werden können ('Stadtgebiet Berlin, nicht zuzuordnen' und 'Berlin (PKS
          gesamt)').
91
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    ```

```

```

131 4          Spandau          25564          4.75
132 5      Steglitz-Zehlendorf      23719          4.40
133 6      Tempelhof-Schöneberg      41776          7.78
134 7          Neukölln          45313          8.45
135 8      Treptow-Köpenick          26099          4.86
136 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 136: API 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
            spalte not in df.columns]

```

```

29         if fehlende_spalten:
30             raise ValueError(f"Die folgenden benötigten Spalten fehlen im
Excel-Sheet: {fehlende_spalten}")
31
32         # Gesamte Straftaten aus der Gesamtzeile holen
33         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 '
79           Fallzahlen_2023' eingelesen.
80         - **Spaltenüberprüfung:** Es wird überprüft, ob die benötigten Spalten ('
81           Bezirke' und 'Straftaten_insgesamt') vorhanden sind.
82         - **Gesamtsumme ermitteln:** Die Gesamtsumme der Straftaten wird entweder
83           aus der Zeile 'Berlin (PKS gesamt)' entnommen oder durch Summe der
84           Bezirkswerte berechnet, falls die Gesamtzeile nicht vorhanden ist.
85         - **Bezirke filtern:** Bezirke wie 'Berlin (PKS gesamt)' und 'Stadtgebiet
86           Berlin, nicht zuzuordnen' werden aus der Berechnung ausgeschlossen.
87         - **Prozentualen Anteil berechnen:** Für jeden Bezirk wird der prozentuale
88           Anteil der Straftaten berechnet und auf zwei Dezimalstellen gerundet.
89         - **Ergebnis anzeigen und speichern:** Das Ergebnis wird sowohl im
90           Terminal angezeigt als auch optional in eine neue Excel-Datei '
91           Straftaten_Prozentual.xlsx' gespeichert.
92
93     3. **Fehlerbehandlung:**
94         - Das Skript enthält Fehlerbehandlungen für fehlende Dateien, fehlende
95           Spalten und unerwartete Fehler, die hilfreiche Fehlermeldungen ausgeben.
96
97     ### Ausführen des Skripts
98
99     1. **Speichere das Skript:**
100         - Speichere das obige Skript in einer Datei, zum Beispiel '
101           berechne_straftaten.py'.
102
103     2. **Führe das Skript aus:**
104         - Öffne das Terminal oder die Eingabeaufforderung.
105         - Navigiere zum Verzeichnis, in dem sich das Skript und die Excel-Datei
106           befinden.
107         - Führe das Skript mit dem folgenden Befehl aus:
108
109         '''bash
110         python berechne_straftaten.py
111         '''
112
113     3. **Ergebnis:**
114         - Das Skript zeigt den prozentualen Anteil der Straftaten pro Bezirk im
115           Terminal an.
116         - Zusätzlich wird eine neue Excel-Datei 'Straftaten_Prozentual.xlsx'
117           erstellt, die diese Informationen enthält.
118
119     ### Beispielausgabe

```



```

108 '''
109 Gesamte Straftaten in Berlin: 536695
110
111 Prozentualer Anteil der Straftaten pro Bezirk:
112
113      Bezirke  Straftaten_insgesamt  Prozentualer_Anteil
114      (%)
115 0           Mitte                    83611             15.57
116 1   Friedrichshain-Kreuzberg          55838
117      10.40
118 2           Pankow                    39164             7.30
119 3   Charlottenburg-Wilmersdorf        53630
120      10.00
121 4           Spandau                    25564             4.77
122 5   Steglitz-Zehlendorf                23719             4.42
123 6   Tempelhof-Schöneberg              41776             7.78
124 7           Neukölln                  45313             8.46
125 8   Treptow-Köpenick                  26099             4.86
126 9   Marzahn-Hellersdorf               26649             4.97
127 10          Lichtenberg               31563             5.89
128 11         Reinickendorf              27344             5.10
129
130 Das Ergebnis wurde in 'Straftaten_Prozentual.xlsx' gespeichert.
131 '''
132
133 ### Hinweise
134
135 - **Dateipfade:** Stelle sicher, dass sich die Datei 'Fallzahlen.xlsx' im
    gleichen Verzeichnis wie das Skript befindet oder gib den vollständigen
    Pfad zur Datei an.
136
137 - **Excel-Format:** Das Skript geht davon aus, dass die Excel-Datei das
    richtige Format hat und die genannten Spalten enthält.
138
139 - **Erweiterungen:** Du kannst das Skript anpassen, um weitere Analysen
    durchzuführen oder die Ausgabe an deine Bedürfnisse anzupassen.
140
141 Ich hoffe, dieses Skript hilft dir weiter! Wenn du Fragen hast oder weitere
    Anpassungen benötigst, stehe ich gerne zur Verfügung.

```

Listing 137: API 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 ### Erklärung des Skripts

```

```

58
59 1. **Bibliotheken importieren:**
60   - 'pandas' wird verwendet, um die Excel-Datei zu lesen und zu verarbeiten.
61
62 2. **Excel-Datei lesen:**
63   - Das Skript liest die Excel-Datei 'Fallzahlen.xlsx' und das spezifische
64     Sheet 'Fallzahlen_2023'.
65
66 3. **Validierung:**
67   - Es überprüft, ob die erforderlichen Spalten ('Bezirke' und '
68     Straftaten_insgesamt') vorhanden sind.
69   - Es stellt sicher, dass die Gesamtzeile 'Berlin (PKS gesamt)' existiert
70     .
71
72 4. **Berechnung der Prozentwerte:**
73   - Die Gesamtanzahl der Straftaten in ganz Berlin ('gesamt_straftaten')
74     wird extrahiert.
75   - Die Gesamtzeile wird aus dem DataFrame entfernt, sodass nur die
76     einzelnen Bezirke übrig bleiben.
77   - Für jeden Bezirk wird der prozentuale Anteil der 'Straftaten_insgesamt'
78     berechnet.
79
80 5. **Ausgabe und Speicherung:**
81   - Die Ergebnisse werden im Terminal angezeigt.
82   - Optional können die Ergebnisse in eine neue Excel-Datei '
83     Fallzahlen_mit_Prozentsatz.xlsx' gespeichert werden.
84
85 ### Beispielausgabe
86
87 Nach dem Ausführen des Skripts erhalten Sie eine Ausgabe ähnlich der
88 folgenden:
89
90 '''
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

	Bezirke	Straftaten_insgesamt	Prozent_Straftaten_insgesamt
0	Mitte	83611	15.58
1	Friedrichshain-Kreuzberg	55838	10.40
2	Pankow	39164	7.28
3	Charlottenburg-Wilmersdorf	53630	9.99
4	Spandau	25564	4.77
5	Steglitz-Zehlendorf	23719	4.43
6	Tempelhof-Schöneberg	41776	7.79
7	Neukölln	45313	8.47
8	Treptow-Köpenick	26099	4.87
9	Marzahn-Hellersdorf	26649	4.97
10	Lichtenberg	31563	5.89

```

95 11          Reinickendorf          27344
    5.10
96 12 Stadtgebiet Berlin, nicht zuzuordnen          56425
    10.52
97
98 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 138: API 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

```

```

= False)
72     print(f"\nDie Ergebnisse wurden in '{output_datei}' gespeichert.")
73 except Exception as e:
74     print(f"Fehler beim Speichern der Datei: {e}")
75 '''
76
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

```

	Bezirke	Straftaten_insgesamt	Prozentualer_Anteil
(%)			
0	Mitte	83611	15.58
1	Friedrichshain-Kreuzberg	55838	
10.40			
2	Pankow	39164	7.31

```

112 3      Charlottenburg-Wilmersdorf      53630
      10.00
113 4      Spandau      25564      4.76
114 5      Steglitz-Zehlendorf      23719      4.42
115 6      Tempelhof-Schöneberg      41776      7.78
116 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 139: API 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
    (2)

```



```

60 # Anzeigen der Ergebnisse
61 print(bezirke_df[['Bezirke', 'Straftaten_insgesamt', 'Prozentualer_Anteil']])
62
63 # Speichern der Ergebnisse in einer neuen Excel-Datei
64 ausgabe_datei = 'Fallzahlen_Prozente.xlsx'
65 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 **Erklärung der wichtigsten Teile des Skripts:**
74
75 - **Lesen der Excel-Datei:**
76     '''python
77     df = pd.read_excel(excel_datei, sheet_name=sheet_name)
78     '''
79     Dies liest das angegebene Sheet der Excel-Datei in ein DataFrame.
80
81 - **Identifizieren des Gesamtwerts:**
82     '''python
83     gesamt_zeile = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
84     gesamt_straftaten = gesamt_zeile['Straftaten_insgesamt'].values[0]
85     '''
86     Hier wird die Zeile gefunden, die den Gesamtwert für Berlin enthält.
87
88 - **Berechnen des prozentualen Anteils:**
89     '''python
90     bezirke_df['Prozentualer_Anteil'] = (bezirke_df['Straftaten_insgesamt'] /
91     gesamt_straftaten) * 100
92     bezirke_df['Prozentualer_Anteil'] = bezirke_df['Prozentualer_Anteil'].round
93     (2)
94     '''
95     Diese Zeilen berechnen den prozentualen Anteil und runden ihn auf zwei
96     Dezimalstellen.
97
98 - **Speichern der Ergebnisse:**
99     '''python
100     bezirke_df.to_excel(ausgabe_datei, sheet_name='Prozentuale_Fallzahlen',
101                        index=False)
102     '''
103     Die Ergebnisse werden in eine neue Excel-Datei mit einem neuen Sheet
104     gespeichert.
105
106 **Ausgabe:**
107
108 Das Skript gibt die Bezirke zusammen mit den 'Straftaten_insgesamt' und ihrem
109 prozentualen Anteil aus und speichert die Ergebnisse in der Datei '
110 Fallzahlen_Prozente.xlsx'.

```

```

103
104 **Beispielausgabe:**
105 '''
106             Bezirke   Straftaten_insgesamt   Prozentualer_Anteil
107 0             Mitte             83611             15.58
108 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 140: API 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
27 nicht gefunden.")
28
29
30 berlin_gesamt = berlin_gesamtreihe['Straftaten_insgesamt'].iloc[0]
31
32 # Entferne die Gesamtzeile aus dem DataFrame
33 bezirke_df = df[df['Bezirke'] != 'Berlin (PKS gesamt)'].copy()
34
35 # Berechne den prozentualen Anteil der Straftaten für jeden Bezirk
36 bezirke_df['Prozentualer_Anteil'] = ((bezirke_df['Straftaten_insgesamt'] /
37     berlin_gesamt) * 100).round(2)
38
39 # Ausgabe der Ergebnisse
40 print(bezirke_df[['Bezirke', 'Straftaten_insgesamt', 'Prozentualer_Anteil']])
41
42
43 # Speichere das Ergebnis in eine neue Excel-Datei
44 bezirke_df.to_excel('Fallzahlen_mit_Prozentanteilen.xlsx', index=False)
45
46
47 ## Erklärung des Skripts
48
49
50 1. **Import der Bibliothek:**
51     '''python
52     import pandas as pd
53     '''
54
55
56 2. **Definition von Datei- und Blattnamen:**
57     '''python
58     excel_datei = 'Fallzahlen.xlsx'
59     blatt_name = 'Fallzahlen_2023'
60     '''
61
62
63 3. **Laden der Excel-Daten:**
64     '''python
65     df = pd.read_excel(excel_datei, sheet_name=blatt_name)
66     '''
67
68     Dies liest die Daten aus dem angegebenen Sheet in ein Pandas DataFrame ein
69     .
70
71
72 4. **Ermittlung der Gesamtanzahl der Straftaten für ganz Berlin:**
73     '''python
74     berlin_gesamtreihe = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
75     if berlin_gesamtreihe.empty:
76         raise ValueError("Die Zeile 'Berlin (PKS gesamt)' wurde in den Daten
77 nicht gefunden.")

```

```

67
68 berlin_gesamt = berlin_gesamtreihe['Straftaten_insgesamt'].iloc[0]
69 '''
70 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
117           Bezirke  Straftaten_insgesamt  Prozentualer_Anteil
118 0           Mitte                83611                15.58
119 1  Friedrichshain-Kreuzberg                55838                10.41
120 2           Pankow                39164                7.30
121 3  Charlottenburg-Wilmersdorf                53630                10.00
122 4           Spandau                25564                4.77
123 5  Steglitz-Zehlendorf                23719                4.42
124 6  Tempelhof-Schöneberg                41776                7.79
125 7           Neukölln                45313                8.48
126 8  Treptow-Köpenick                26099                4.87
127 9  Marzahn-Hellersdorf                26649                4.97
128 10          Lichtenberg                31563                5.89
129 11          Reinickendorf                27344                5.10
130 12 Stadtgebiet Berlin, nicht zuzuordnen                56425
    10.54
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 141: API 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.

```

```

54
55 5. **Berechnung des prozentualen Anteils:**
56 - Für jeden Bezirk wird der prozentuale Anteil der Straftaten berechnet,
    indem die Straftaten des Bezirks durch die Gesamtzahl der Straftaten in
    Berlin geteilt und mit 100 multipliziert wird.
57
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
85      10.532238
86
87 Die Ergebnisse wurden in 'Fallzahlen_mit_Prozent.xlsx' gespeichert.
88 '''
89
90 ### Hinweise
91
92 - **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.
93
94 - **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 - **Anpassungen:** Du kannst das Skript nach Bedarf erweitern, z.B. durch
    Visualisierungen oder weitere Analysen.
95
96 Wenn du Fragen hast oder weitere Anpassungen benötigst, stehe ich gerne zur
    Verfügung!

```

Listing 142: API 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         ")
26         return
27
28     # 2. Tabelle als Pandas DataFrame speichern
29     # (bereits durch das Einlesen mit pandas erledigt)
30
31     # 3. Überprüfen, ob die notwendigen Spalten vorhanden sind
32     erforderliche_spalten = ['Bezirke', 'Straftaten_insgesamt']
33     fehlende_spalten = [spalte for spalte in erforderliche_spalten if spalte
34                          not in df.columns]
35
36     if fehlende_spalten:
37         print(f"Fehler: Fehlende Spalten in der Excel-Datei: {
38         fehlende_spalten}")
39         return
40     else:
41         print("Alle erforderlichen Spalten sind vorhanden.")

```



```

39
40 # 4. Gesamtzahl der Straftaten für ganz Berlin finden
41 gesamt_row = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
42
43 if gesamt_row.empty:
44     print("Fehler: Die Zeile mit 'Berlin (PKS gesamt)' wurde nicht
45         gefunden.")
46     return
47
48 gesamt_straftaten = gesamt_row['Straftaten_insgesamt'].values[0]
49 print(f"Gesamtzahl der Straftaten für Berlin: {gesamt_straftaten}")
50
51 # 5. Prozentualen Anteil der einzelnen Bezirke berechnen
52 # Ausschließen der Gesamtzeile für die Berechnung
53 df_bezirke = df[df['Bezirke'] != 'Berlin (PKS gesamt)'].copy()
54
55 if df_bezirke.empty:
56     print("Fehler: Es gibt keine Bezirke nach dem Ausschluss von 'Berlin
57         (PKS gesamt)'")
58     return
59
60 # Prozentualer Anteil berechnen
61 df_bezirke['Prozentualer_Anteil (%)'] = (df_bezirke['Straftaten_insgesamt']
62     / gesamt_straftaten) * 100
63
64 # Optional: Runden auf zwei Dezimalstellen
65 df_bezirke['Prozentualer_Anteil (%)'] = df_bezirke['Prozentualer_Anteil
66     (%)'].round(2)
67
68 print("Prozentualer Anteil der Straftaten pro Bezirk berechnet.")
69
70 # 6. Ergebnis als DataFrame zurückgeben
71 # Hier zeigen wir das Ergebnis an. Sie können es auch in eine Datei
72 speichern.
73 ergebnis_df = df_bezirke[['Bezirke', 'Straftaten_insgesamt', '
74     Prozentualer_Anteil (%)']]
75 print("\nErgebnis DataFrame:")
76 print(ergebnis_df.to_string(index=False))
77
78 # Optional: Ergebnis in eine neue Excel-Datei speichern
79 ausgabe_datei = 'Fallzahlen_prozentual.xlsx'
80 try:
81     ergebnis_df.to_excel(ausgabe_datei, index=False)
82     print(f"\nErgebnis erfolgreich in '{ausgabe_datei}' gespeichert.")
83 except Exception as e:
84     print(f"Ein Fehler ist beim Speichern der Ergebnisdatei aufgetreten:
85         {e}")
86
87 if __name__ == "__main__":
88     main()
89
90
91

```

```

84 ### Erklärung der Schritte
85
86 1. **Excel-Datei einlesen**:
87     - Die Excel-Datei 'Fallzahlen.xlsx' wird mit der Methode 'pd.read_excel'
      eingelesen.
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'
      im DataFrame vorhanden sind.
96     - 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'
133       erstellt, die das Ergebnis enthält.
134
135 134 ### Beispielausgabe
136
137     '''
138     Excel-Datei 'Fallzahlen.xlsx' erfolgreich eingelesen.
139     Alle erforderlichen Spalten sind vorhanden.
140     Gesamtzahl der Straftaten für Berlin: 536695
141     Prozentualer Anteil der Straftaten pro Bezirk berechnet.
142
143     Ergebnis DataFrame:
144
145         Bezirke   Straftaten_insgesamt   Prozentualer_Anteil (%)
146         Mitte           83611           15.57
147         Friedrichshain-Kreuzberg           55838           10.41
148         Pankow           39164           7.30
149         Charlottenburg-Wilmersdorf           53630           10.00
150         Spandau           25564           4.77
151         Steglitz-Zehlendorf           23719           4.42
152         Tempelhof-Schöneberg           41776           7.78
153         Neukölln           45313           8.46
154         Treptow-Köpenick           26099           4.87
155         Marzahn-Hellersdorf           26649           4.97
156         Lichtenberg           31563           5.88
157         Reinickendorf           27344           5.10
158         Stadtgebiet Berlin, nicht zuzuordnen           56425
159         10.52
160
161     Ergebnis erfolgreich in 'Fallzahlen_prozentual.xlsx' gespeichert.
162     '''
163
164     Dieses Skript bietet eine robuste Lösung zur Verarbeitung Ihrer Excel-Daten
165     und zur Berechnung der prozentualen Anteile der Straftaten pro Bezirk in
166     Berlin. Sie können das Skript nach Bedarf erweitern oder anpassen, zum
167     Beispiel indem Sie zusätzliche Analysen hinzufügen oder das Ausgabeformat
168     ändern.

```

Listing 143: API 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 prozentualem
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]

```

```

48     print(f"Gesamtzahl der Straftaten in Berlin: {gesamt_straftaten}")
49
50     if gesamt_straftaten == 0:
51         raise ValueError("Die Gesamtzahl der Straftaten ist 0, kann keine
52         Prozente berechnen.")
53
54     # 5. Prozentualen Anteil der einzelnen Bezirke berechnen
55     # Ausschließen der Gesamtzeile von den einzelnen Bezirken
56     einzel_bezirke = df[df['Bezirke'] != 'Berlin (PKS gesamt)'].copy()
57
58     # Berechnung des prozentualen Anteils
59     einzel_bezirke['Prozentualer_Anteil'] = (einzel_bezirke['
60     Straftaten_insgesamt'] / gesamt_straftaten) * 100
61
62     # Optional: Auf zwei Dezimalstellen runden
63     einzel_bezirke['Prozentualer_Anteil'] = einzel_bezirke['
64     Prozentualer_Anteil'].round(2)
65
66     # 6. Ergebnis als DataFrame zurückgeben
67     ergebnis = einzel_bezirke[['Bezirke', 'Straftaten_insgesamt', '
68     Prozentualer_Anteil']]
69
70     return ergebnis
71
72 if __name__ == "__main__":
73     dateipfad = 'Fallzahlen.xlsx' # Pfad zur Excel-Datei anpassen, falls
74     erforderlich
75     try:
76         ergebnis_df = verarbeite_fallzahlen(dateipfad)
77         print("\nProzentualer Anteil der Straftaten pro Bezirk in Berlin:")
78         print(ergebnis_df.to_string(index=False))
79     except Exception as e:
80         print(f"Ein Fehler ist aufgetreten: {e}")
81
82 '''
83
84 ### Erklärung der Schritte
85
86 1. **Excel-Datei einlesen:**
87     - Das Skript verwendet 'pd.read_excel', um die Excel-Datei mit dem
88     angegebenen Sheet-Namen ('Fallzahlen_2023') einzulesen.
89     - Fehler beim Einlesen der Datei werden abgefangen und eine entsprechende
90     Fehlermeldung ausgegeben.
91
92 2. **Überprüfen der notwendigen Spalten:**
93     - Es werden überprüft, ob die Spalten 'Bezirke' und '
94     Straftaten_insgesamt' im DataFrame vorhanden sind.
95     - Falls eine oder beide Spalten fehlen, wird eine Fehlermeldung ausgegeben
96     .
97
98 3. **Gesamtzahl der Straftaten finden:**
99     - Die Zeile mit dem Bezirk 'Berlin (PKS gesamt)' wird gefunden, um die
100     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 Dieses Ergebnis zeigt den prozentualen Anteil der Straftaten für jeden Bezirk
    im Vergleich zur Gesamtzahl der Straftaten in Berlin.

```

Listing 144: API 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:
"""
Liest eine Excel-Datei ein und berechnet den prozentualen Anteil der
Straftaten_insgesamt für jeden Bezirk im Vergleich zu ganz Berlin.

Parameters:
- excel_file: Pfad zur Excel-Datei.
- sheet_name: Name des Sheets in der Excel-Datei (standardmäßig '
Fallzahlen_2023').

Returns:
- Ein Pandas DataFrame mit den Bezirken und ihrem prozentualen Anteil.
"""

try:
    # 1. Excel-Datei einlesen
    df = pd.read_excel(excel_file, sheet_name=sheet_name)
    print("Excel-Datei erfolgreich eingelesen.")
except FileNotFoundError:
    raise FileNotFoundError(f"Die Datei '{excel_file}' wurde nicht
gefunden.")
except Exception as e:
    raise Exception(f"Fehler beim Einlesen der Excel-Datei: {e}")

# 2. DataFrame gespeichert (bereits in df)

# 3. Überprüfen, ob die notwendigen Spalten vorhanden sind
required_columns = ['Bezirke', 'Straftaten_insgesamt']
missing_columns = [col for col in required_columns if col not in df.
columns]
if missing_columns:
    raise ValueError(f"Die folgenden erforderlichen Spalten fehlen: {'',
'.join(missing_columns)}")
print("Alle erforderlichen Spalten sind vorhanden.")

# 4. Gesamtzahl der Straftaten für ganz Berlin finden
total_row = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
if total_row.empty:
    raise ValueError("Die Zeile mit 'Berlin (PKS gesamt)' wurde nicht
gefunden.")
total_straftaten = total_row['Straftaten_insgesamt'].values[0]
print(f"Gesamtzahl der Straftaten für ganz Berlin: {total_straftaten}")

# 5. Prozentualen Anteil der einzelnen Bezirke berechnen
# Ausschließen der Gesamtzeile aus den Bezirken
bezirk_df = df[df['Bezirke'] != 'Berlin (PKS gesamt)'].copy()

# Berechnung des prozentualen Anteils
bezirk_df['Prozentualer_Anteil'] = (bezirk_df['Straftaten_insgesamt'] /
total_straftaten) * 100

# 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

```



```

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

```

```

56
57 2. **Überprüfung der notwendigen Spalten**:
58 - Es wird geprüft, ob die Spalten 'Bezirke' und 'Straftaten_insgesamt' im
59 DataFrame vorhanden sind.
60 - Falls eine der Spalten fehlt, wird eine Fehlermeldung ausgegeben und das
61 Skript beendet.
62
63 3. **Gesamtzahl der Straftaten bestimmen**:
64 - Die Zeile mit dem Bezirk 'Berlin (PKS gesamt)' wird gesucht, um die
65 Gesamtzahl der Straftaten in ganz Berlin zu erhalten.
66 - Falls diese Zeile nicht gefunden wird, wird eine Fehlermeldung
67 ausgegeben.
68
69 4. **Prozentualen Anteil berechnen**:
70 - Die Gesamtzeile wird aus dem DataFrame entfernt, um nur die einzelnen
71 Bezirke zu betrachten.
72 - Für jeden Bezirk wird der prozentuale Anteil der Straftaten am
73 Gesamtwert berechnet.
74 - Optional wird der prozentuale Anteil auf zwei Dezimalstellen gerundet.
75
76 5. **Ergebnis darstellen**:
77 - Es wird ein neues DataFrame erstellt, das die Bezirke, die Gesamtzahl
78 der Straftaten und den prozentualen Anteil enthält.
79 - Das Ergebnis wird in der Konsole ausgegeben.
80
81 ### Beispielausgabe:
82
83 '''
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

	Bezirke	Straftaten_insgesamt	Prozentualer_Anteil
0	Mitte	83611	15.60
1	Friedrichshain-Kreuzberg	55838	10.40
2	Pankow	39164	7.30
3	Charlottenburg-Wilmersdorf	53630	10.00
4	Spandau	25564	4.80
5	Steglitz-Zehlendorf	23719	4.40
6	Tempelhof-Schöneberg	41776	7.80
7	Neukölln	45313	8.50
8	Treptow-Köpenick	26099	4.90
9	Marzahn-Hellersdorf	26649	5.00
10	Lichtenberg	31563	5.90
11	Reinickendorf	27344	5.10
12	Stadtgebiet Berlin, nicht zuzuordnen	56425	
	10.50		

```

1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500

```

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 146: API 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 prozentualem 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.

```

```

89 - Der prozentuale Anteil wird für jeden Bezirk berechnet und als neue
    Spalte 'Prozentualer_Anteil' zum DataFrame hinzugefügt.
90
91 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 147: API Response Testfall 3 Prompt 3 Ausführung 5

11.3.11 Komplette Antworten der API Requests Testfall 4

```

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
10    ", "2021".
11    """
12    import re
13    match = re.search(r'(\d{4})', sheet_name)
14    if match:
15        return int(match.group(1))
16    else:
17        raise ValueError(f"Kein Jahr im Sheetnamen '{sheet_name}' gefunden.")
18
19 def get_total_crimes(df):
20     """
21     Extrahiert die Gesamtzahl der Straftaten für Berlin aus dem DataFrame.
22     Annahme: Die Zeile mit 'Bezirke' == 'Berlin (PKS gesamt)' enthält die
23     Gesamtzahl.
24     """
25     total_row = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
26     if total_row.empty:
27         raise ValueError("Keine Zeile mit 'Berlin (PKS gesamt)' gefunden.")
28     return total_row['Straftaten_insgesamt'].values[0]
29
30 def main():
31     # Pfad zur Excel-Datei
32     excel_file = 'Fallzahlen.xlsx'
33
34     # Lade die Excel-Datei
35     try:
36         xls = pd.ExcelFile(excel_file)
37     except FileNotFoundError:
38         print(f"Datei '{excel_file}' wurde nicht gefunden.")
39         return
40     except Exception as e:
41         print(f"Fehler beim Laden der Datei: {e}")
42         return
43
44     # Liste der Sheets
45     sheet_names = xls.sheet_names
46
47     # Liste zur Speicherung der Ergebnisse
48     data = []
49
50     for sheet in sheet_names:
51         try:
52             # Lade das Sheet als DataFrame
53             df = pd.read_excel(xls, sheet_name=sheet)
54
55             # Extrahiere das Jahr

```

```

54         year = extract_year(sheet)
55
56         # Extrahiere die Gesamtzahl der Straftaten
57         total_crimes = get_total_crimes(df)
58
59         # Speichere das Ergebnis
60         data.append({'Jahr': year, 'Straftaten_insgesamt': total_crimes})
61
62     except Exception as e:
63         print(f"Fehler beim Verarbeiten des Sheets '{sheet}': {e}")
64         continue
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.

```



```

100 - 'get_total_crimes(df)': Diese Funktion filtert die Zeile, in der die
    Spalte 'Bezirke' den Wert 'Berlin (PKS gesamt)' hat, und gibt den Wert
    aus der Spalte 'Straftaten_insgesamt' zurück.
101
102 3. **Hauptfunktion ('main'):**
103 - Lädt die Excel-Datei 'Fallzahlen.xlsx'.
104 - Iteriert über alle Sheets in der Excel-Datei:
105     - 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 148: API 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 jä
  hrliche Gesamtstraftatenzahl 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:

```

```

47     # Lade alle Sheets der Excel-Datei
48     xls = pd.ExcelFile(excel_file)
49 except FileNotFoundError:
50     print(f"Die Datei {excel_file} wurde nicht gefunden.")
51     return
52 except Exception as e:
53     print(f"Fehler beim Laden der Excel-Datei: {e}")
54     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 except Exception as e:
110     print(f"Fehler beim Speichern der Ergebnisdatei: {e}")
111
112 if __name__ == "__main__":
113     main()
114 '''
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
der Straftaten für ganz Berlin.
123     - Sie sucht zuerst nach dem 'LOR-Schlüssel' '999999'. Wenn dieser nicht
gefunden wird, sucht sie nach dem Bezirksnamen 'Berlin (PKS gesamt)'.
124     - Wenn beide Suchen fehlschlagen, gibt sie 'NaN' zurück.
125
126 3. **Funktion 'main'**:
127     - Lädt die Excel-Datei und iteriert über alle Sheets.
128     - Extrahiert für jedes Sheet die Gesamtzahl der Straftaten in Berlin.
129     - Versucht, das Jahr aus dem Sheet-Namen zu extrahieren. Wenn der Sheet-
Name kein gültiges Jahr ist, wird das Sheet übersprungen.
130     - Speichert die gesammelten Daten in einem DataFrame.
131     - Sortiert die Daten nach Jahr und berechnet die prozentuale Veränderung
zum Vorjahr.
132     - Gibt den resultierenden DataFrame aus und speichert ihn optional in
eine neue Excel-Datei namens 'Straftaten_Zeitreihe.xlsx'.
133
134 4. **Ausführen des Skripts**:
135     - Das Skript kann direkt ausgeführt werden, indem du es speicherst (z.B.
als 'analyse_fallzahlen.py') und über die Kommandozeile startest:
136     ```bash
137     python analyse_fallzahlen.py
138     ```
139
140 ### Beispielausgabe

```

```

141
142 Nach dem Ausführen des Skripts erhältst du einen DataFrame ähnlich dem
    folgenden:
143
144 '''
145      Straftaten_insgesamt  Prozentuale_Veraenderung
146 Jahr
147 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 149: API 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 import pandas as pd
6
7 # Pfad zur Excel-Datei
8 excel_datei = 'Fallzahlen.xlsx'
9
10 # Lade alle Sheets in ein Dictionary
11 alle_sheets = pd.read_excel(excel_datei, sheet_name=None)
12
13 # Liste zur Speicherung der jährlichen Straftaten

```

```

13 jährliche_straftaten = []
14
15 for sheet_name, df in alle_sheets.items():
16     try:
17         # Versuche, das Jahr aus dem Sheetnamen zu extrahieren
18         jahr = int(sheet_name)
19     except ValueError:
20         # 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'
: gesamt_straftaten})
80         else:
81             print(f"Keine Daten für 'Berlin (PKS gesamt)' in Sheet '{
sheet_name}' gefunden.")
82     '''
83     - Es wird angenommen, dass jeder Sheetname ein Jahr darstellt (z.B. '2020
', '2021', etc.).
84     - Das Skript sucht in jedem Sheet nach der Zeile, in der die Bezirke '
Berlin (PKS gesamt)' sind, und extrahiert die Gesamtzahl der Straftaten.
85
86 4. **Erstellen des DataFrames**:
87     '''python
88     df_jährlich = pd.DataFrame(jährliche_straftaten)
89     df_jährlich = df_jährlich.sort_values('Jahr').reset_index(drop=True)
90     '''
91     Die gesammelten Daten werden in einen DataFrame umgewandelt und nach Jahr
sortiert.
92
93 5. **Berechnung der prozentualen Veränderung**:
94     '''python
95     df_jährlich['Veränderung_prozent'] = df_jährlich['Straftaten_insgesamt'].
pct_change() * 100
96     df_jährlich['Veränderung_prozent'] = df_jährlich['Veränderung_prozent'].
round(2)
97     '''
98     - 'pct_change()' berechnet die prozentuale Veränderung zum Vorjahr.
99     - Die Ergebnisse werden auf zwei Dezimalstellen gerundet.
100
101 6. **Ausgabe des Ergebnisses**:
102     '''python
103     print(df_jährlich)
104     '''

```

```

104     Dies gibt den finalen DataFrame mit den jährlichen Straftaten und der
105     prozentualen Veränderung zum Vorjahr aus.
106
107     ### Beispielhafte Ausgabe
108
109     Angenommen, deine Excel-Datei enthält die Daten für die Jahre 2018 bis 2022,
110     könnte die Ausgabe wie folgt aussehen:
111
112     '''
113
114         Jahr    Straftaten_insgesamt    Veränderung_prozent
115     0    2018                50000                NaN
116     1    2019                52000                4.00
117     2    2020                51000               -1.92
118     3    2021                53000                3.92
119     4    2022                55000                3.77
120     '''
121
122     - **Hinweis**: Für das erste Jahr ('2018' in diesem Beispiel) gibt es keine
123       Veränderung zum Vorjahr, daher ist der Wert 'NaN'.
124
125     ### Anpassungen
126
127     Falls deine Sheets nicht direkt das Jahr im Namen enthalten, sondern ein
128     anderes Format haben (z.B. 'Daten_2020'), musst du die Methode zur
129     Extraktion des Jahres aus dem Sheetnamen anpassen. Hier ein Beispiel, wie
130     du das Jahr extrahieren könntest, wenn der Sheetname das Jahr enthält:
131
132     '''python
133     import re
134
135     # Beispiel: Sheetname ist 'Daten_2020'
136     match = re.search(r'(\d{4})', sheet_name)
137     if match:
138         jahr = int(match.group(1))
139     else:
140         print(f"Jahr in Sheet '{sheet_name}' nicht gefunden.")
141         continue
142     '''
143
144     Dieses Snippet verwendet reguläre Ausdrücke, um eine vierstellige Jahreszahl
145     im Sheetnamen zu finden.
146
147     ### Fehlerbehandlung
148
149     Das Skript enthält grundlegende Fehlerbehandlungen, wie das Überspringen von
150     Sheets, die kein gültiges Jahr als Namen haben, und das Informieren, wenn
151     die Zeile für 'Berlin (PKS gesamt)' nicht gefunden wird. Du kannst diese
152     erweitern, um spezifischere Anforderungen oder Datenstrukturen zu unterstüt-
153     zen.
154
155     ### Abhängigkeiten

```



```

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 150: API 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
sentierte 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.

```

```

112 ### Beispiel-Ausgabe:
113
114 Angenommen, die Excel-Datei enthält Daten für die Jahre 2018 bis 2023, könnte
    die Ausgabe wie folgt aussehen:
115
116 '''
117      Straftaten_insgesamt   Prozentuale_Veraenderung_zum_Vorjahr (%)
118 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 151: API 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'].

```

```

round(2)
'''
119
120
121 8. **Anzeige und optionales Speichern des Ergebnisses:**
122     '''python
123     print(df_gesamt)
124     df_gesamt.to_excel('Prozentuale_Veraenderung_Straftaten_Berlin.xlsx')
125     '''
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 152: API 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
bersprungen.")
32         continue
33
34 # Sortiere die Sheets nach Jahr
35 sheet_years_sorted = sorted(sheet_years, key=lambda x: x[0])
36
37 # Liste zur Speicherung der Ergebnisse
38 data = []
39
40 # Iteriere über jedes Sheet und extrahiere die benötigten Daten
41 for year, sheet in sheet_years_sorted:
42     # Lese das aktuelle Sheet
43     df = pd.read_excel(excel_file, sheet_name=sheet)
44
45     # Stelle sicher, dass die benötigten Spalten vorhanden sind
46     if 'Bezirke' not in df.columns or 'Straftaten_insgesamt' not in df.
columns:

```



```

47     print(f"Sheet '{sheet}' fehlt die erforderliche Spalte. Überspringe
dieses Sheet.")
48     continue
49
50     # Filtere die Zeile für "Berlin (PKS gesamt)"
51     row = df[df['Bezirke'] == "Berlin (PKS gesamt)"]
52
53     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
NaN.")
60     else:
61         print(f"'Berlin (PKS gesamt)' nicht in Sheet '{sheet}' gefunden.")
62
63 # Erstelle ein DataFrame aus den gesammelten Daten
64 df_data = pd.DataFrame(data)
65
66 # Sortiere das DataFrame nach Jahr, um sicherzustellen, dass die Reihenfolge
stimmt
67 df_data = df_data.sort_values('Year').reset_index(drop=True)
68
69 # Berechne die prozentuale Veränderung zum Vorjahr
70 df_data['Prozentuale_Veraenderung'] = df_data['Straftaten_insgesamt'].
pct_change() * 100
71
72 # Optional: Runde die Prozentwerte auf zwei Dezimalstellen
73 df_data['Prozentuale_Veraenderung'] = df_data['Prozentuale_Veraenderung'].
round(2)
74
75 # Anzeige des Ergebnisses
76 print(df_data)
77
78 # Optional: Speichere das Ergebnis in eine neue Excel-Datei oder CSV
79 # df_data.to_excel('Prozentuale_Veraenderung_Fallzahlen.xlsx', index=False)
80 # df_data.to_csv('Prozentuale_Veraenderung_Fallzahlen.csv', index=False)
81 '''
82
83 ## Erklärung der einzelnen Schritte
84
85 1. **Importieren der Bibliotheken**:
86     - 'pandas' wird für die Datenverarbeitung verwendet.
87
88 2. **Laden der Excel-Datei**:
89     - Mit 'pd.ExcelFile' werden alle Sheet-Namen aus der Excel-Datei
extrahiert.
90
91 3. **Extrahieren und Sortieren der Jahre**:
92     - Die Sheet-Namen werden erwartet im Format 'Fallzahlen_Jahr' (z.B. '

```

```

Fallzahlen_2014').
93 - Das Jahr wird extrahiert und die Sheets werden in aufsteigender
    Reihenfolge sortiert.
94
95 4. **Datenextraktion je Sheet**:
96 - Für jedes sortierte Sheet wird das entsprechende DataFrame geladen.
97 - 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 zusä
    tzliche 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 153: API 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 Verä
  nderung 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,
15                               sheet_prefix):
16     """
17     Liest eine Excel-Datei mit mehreren Sheets, extrahiert die
18     Straftaten_insgesamt für
19     einen bestimmten Eintrag und berechnet die prozentuale Veränderung zum
20     Vorjahr.
21
22     Parameters:
23     - excel_file: str, Pfad zur Excel-Datei.
24     - target_row: str, Der Eintrag in der 'Bezirke' Spalte, z.B. "Berlin (PKS
25       gesamt)".
26     - target_column: str, Name der Spalte, deren prozentuale Veränderung
27       berechnet werden soll, z.B. "Straftaten_insgesamt".
28     - sheet_prefix: str, Präfix der Sheet-Namen, z.B. "Fallzahlen_" für
29       Sheets wie "Fallzahlen_2014".
30
31     Returns:
32     - pandas.DataFrame mit den Jahren und der prozentualen Veränderung.
33     """
34     try:
35         # Alle Sheets einlesen
36         sheets_dict = pd.read_excel(excel_file, sheet_name=None, engine='
37 openpyxl')
38     except FileNotFoundError:
39         print(f"Die Datei {excel_file} wurde nicht gefunden.")
40         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).

```

```

125
126 ### Beispielausgabe
127
128 Angenommen, die Excel-Datei enthält Daten von 2014 bis 2023, könnte die
    Ausgabe wie folgt aussehen:
129
130 '''
131 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 154: API 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 155: API 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
13     der prozentualen Veränderung der Straftaten zum jeweiligen Vorjahr.
14
15     Parameters:
16     - file_path: Pfad zur Excel-Datei
17     - target_district: Name des Bezirks, der analysiert werden soll
18     - column_name: Name der Spalte, deren Veränderung berechnet werden soll
19
20     Returns:
21     - pandas DataFrame mit den Jahreszahlen und der prozentualen Veränderung
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         if row.empty:
56             print(f"Der Bezirk '{target_district}' wurde im Sheet '{sheet}'
nicht gefunden. Überspringe dieses Sheet.")
57             continue
58
59         # Extrahiere den Wert der gewünschten Spalte
60         try:
61             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=
104         False)
105
106     """
107
108     ### Erklärung des Skripts:
109
110     1. **Importieren der Bibliothek:**
111     - 'pandas' wird für die Datenmanipulation und -analyse verwendet.
112
113     2. **Funktion 'analyze_fallzahlen':**
114     - **Parameter:**
115       - 'file_path': Der Pfad zur Excel-Datei.
116       - 'target_district': Der Bezirk, der analysiert werden soll (Standard: "
117       Berlin (PKS gesamt)").
118       - 'column_name': Die Spalte, deren Veränderung berechnet werden soll (
119       Standard: "Straftaten_insgesamt").
120     - **Schritte:**
121       - Lädt die Excel-Datei und iteriert über alle Sheets.
122       - Extrahiert das Jahr aus dem Sheet-Namen.
123       - Liest das Sheet in einen DataFrame und sucht die Zeile für den
124       Zielbezirk.
125       - Extrahiert den Wert der gewünschten Spalte und speichert ihn im
126       Dictionary 'data'.
127       - Nach der Iteration erstellt der DataFrame 'df_data' aus dem Dictionary
128       , sortiert nach Jahr.
129       - Berechnet die prozentuale Veränderung zum Vorjahr und fügt sie als
130       neue Spalte hinzu.
131
132     3. **Hauptblock ('if __name__ == "__main__":'):**
133     - Definiert den Pfad zur Excel-Datei.
134     - Ruft die Funktion 'analyze_fallzahlen' auf und speichert das Ergebnis in
135       'result_df'.
136     - Gibt das Ergebnis aus.
137     - Optional kann das Ergebnis in eine neue Excel- oder CSV-Datei
138       gespeichert werden (auskommentiert).
139
140     ### Beispielausgabe:
141
142     Angenommen, die Excel-Datei enthält Daten von 2014 bis 2023, könnte die
143     Ausgabe wie folgt aussehen:
144
145     """
146
147     Year    Straftaten_insgesamt    Percent Change
148     0    2014                500000                NaN
149     1    2015                510000                2.00
150     2    2016                495000               -2.94
151     3    2017                530000                7.07
152     4    2018                550000                3.77
153     5    2019                540000               -1.82
154     6    2020                560000                3.70
155     7    2021                580000                3.57
156     8    2022                570000               -1.72

```

```

145 9      2023      590000      3.51
146 '''
147
148 **Hinweis:**
149 - Der erste Jahreseintrag hat keine prozentuale Veränderung zum Vorjahr und
    zeigt daher 'NaN'.
150 - Stellen Sie sicher, dass alle Sheet-Namen dem Format 'Fallzahlen_Jahr'
    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 156: API 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
    den angegebenen Bezirk.
31     """
32     df = pd.read_excel(excel_file, sheet_name=sheet_name)
33

```

```

34 # Annahme: Die Bezeichnung des Bezirks ist in der Spalte 'Bezirke'
35 row = df[df['Bezirke'] == target_district]
36
37 if not row.empty:
38     return row.iloc[0]['Straftaten_insgesamt']
39 else:
40     print(f"Warnung: Bezirk '{target_district}' nicht in Sheet '{
41 sheet_name}' gefunden.")
42     return None
43
44 def main():
45     excel_file = 'Fallzahlen.xlsx'
46     target_district = 'Berlin (PKS gesamt)'
47
48     # Lade alle Sheet-Namen
49     xls = pd.ExcelFile(excel_file)
50     sheet_names = xls.sheet_names
51
52     # Filtere die relevanten Sheets und extrahiere die Jahre
53     pattern = re.compile(r'^Fallzahlen_(\d{4})$')
54     sheets_with_year = []
55     for sheet in sheet_names:
56         match = pattern.match(sheet)
57         if match:
58             year = int(match.group(1))
59             sheets_with_year.append((year, sheet))
60
61     if not sheets_with_year:
62         print("Keine Sheets im erwarteten Format 'Fallzahlen_YYYY' gefunden."
63 )
64         return
65
66     # Sortiere die Sheets nach Jahr
67     sheets_with_year.sort(key=lambda x: x[0])
68
69     # Extrahiere die 'Straftaten_insgesamt' Werte
70     data = {}
71     for year, sheet in sheets_with_year:
72         value = get_straftaten_insgesamt(excel_file, sheet, target_district)
73         if value is not None:
74             data[year] = value
75
76     # Erstelle ein DataFrame
77     df = pd.DataFrame(list(data.items()), columns=['Jahr', '
78 Straftaten_insgesamt'])
79     df.sort_values('Jahr', inplace=True)
80     df.reset_index(drop=True, inplace=True)
81
82     # Berechne die prozentuale Veränderung zum Vorjahr
83     df['Prozentuale_Veraenderung'] = df['Straftaten_insgesamt'].pct_change()
84     * 100

```

```

82     # Optional: Runde die Prozentwerte auf zwei Dezimalstellen
83     df['Prozentuale_Veraenderung'] = df['Prozentuale_Veraenderung'].round(2)
84
85     print(df)
86
87 if __name__ == "__main__":
88     main()
89 '''
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_strafttaten_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 '''

```

```

127
128 ### Hinweise
129
130 - **Fehlende Daten**: Falls ein bestimmtes Jahr oder ein Sheet nicht den benö-
    tigten Bezirk enthält, gibt das Skript eine Warnung aus und überspringt
    diesen Eintrag.
131 - **Anpassungen**: Je nach tatsächlicher Struktur deiner Excel-Datei (z.B.
    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-
    tigst oder Fragen hast, lass es mich wissen.

```

Listing 157: API 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. '
24         Fallzahlen_2014'
25         jahr = int(sheet.split('_')[-1])
26         sheet_jahre.append((jahr, sheet))
27     except ValueError:
28         print(f"Sheet '{sheet}' hat kein gültiges Jahresformat und wird ü
29         bersprungen.")
30
31 # Sortiere die Sheets nach Jahr
32 sheet_jahre.sort()
33
34 # Lade die DataFrames und speichere sie im Dictionary
35 for jahr, sheet in sheet_jahre:
36     df = pd.read_excel(xls, sheet_name=sheet)
37     dataframes[jahr] = df
38
39 # Schritt 2: Extrahiere 'Straftaten_insgesamt' für 'Berlin (PKS gesamt)'
40 strafaten_gesamt = {}
41
42 for jahr, df in dataframes.items():
43     # Filtern der Zeile 'Berlin (PKS gesamt)'
44     berlin_row = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
45     if not berlin_row.empty:
46         wert = berlin_row.iloc[0]['Straftaten_insgesamt']
47         strafaten_gesamt[jahr] = wert
48     else:
49         print(f"Die Zeile 'Berlin (PKS gesamt)' wurde im Jahr {jahr} nicht
50         gefunden.")
51
52 # Schritt 3: Berechne die prozentuale Veränderung zum Vorjahr
53 jahre = sorted(strafaten_gesamt.keys())
54 prozentuale_veraenderung = {}
55
56 for i in range(1, len(jahre)):
57     aktuelles_jahr = jahre[i]
58     vorheriges_jahr = jahre[i-1]
59     wert_aktuell = strafaten_gesamt[aktuelles_jahr]
60     wert_vorher = strafaten_gesamt[vorheriges_jahr]
61     veraenderung = ((wert_aktuell - wert_vorher) / wert_vorher) * 100
62     prozentuale_veraenderung[aktuelles_jahr] = veraenderung
63
64 # Schritt 4: Ergebnisse in einem neuen DataFrame speichern
65 ergebnisse = pd.DataFrame({
66     'Jahr': list(prozentuale_veraenderung.keys()),
67     'Prozentuale Veränderung (%)': list(prozentuale_veraenderung.values())
68 })

```

```

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     \[
92     \text{Veränderung} = \left( \frac{\text{Wert aktuell}}{\text{Wert vorher}} - 1 \right) \times 100
93     \]
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 158: API 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 159: API 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

```

```

111     - Das Jahr wird aus dem Sheet-Namen extrahiert.
112     - Die Zeile mit 'Bezirke' == 'Berlin (PKS gesamt)' wird gesucht und
      der Wert von 'Straftaten_insgesamt' extrahiert.
113
114     - **Erstellen des Ergebnis-DataFrames:**
115     '''python
116     strafaten_df = pd.DataFrame(list(strafaten_dict.items()), columns=['Jahr
117     ', 'Straftaten_insgesamt'])
118     strafaten_df.sort_values('Jahr', inplace=True)
119     strafaten_df.reset_index(drop=True, inplace=True)
120     '''
121
122     - **Berechnen der prozentualen Veränderung:**
123     '''python
124     strafaten_df['Prozentuale_Veraenderung'] = strafaten_df['
125     Straftaten_insgesamt'].pct_change() * 100
126     strafaten_df['Prozentuale_Veraenderung'] = strafaten_df['
127     Prozentuale_Veraenderung'].round(2)
128     '''
129
130     - **Ergebnisse anzeigen und speichern:**
131     '''python
132     print(strafaten_df)
133     output_file = 'Straftaten_Veraenderung.xlsx'
134     strafaten_df.to_excel(output_file, index=False)
135     print(f"Ergebnisse wurden in '{output_file}' gespeichert.")
136     '''
137
138 3. **Skript ausführen:**
139     '''python
140     if __name__ == "__main__":
141         main()
142     '''
143
144     ### Zusätzliche Hinweise:
145
146     - **Fehlerbehandlung:**
147     - Das Skript gibt eine Warnung aus, wenn ein Sheet-Name nicht dem
      erwarteten Format entspricht oder wenn die spezifische Zeile nicht
      gefunden wird.
148
149     - **Speichern der Ergebnisse:**
150     - Die Ergebnisse werden nicht nur auf dem Bildschirm angezeigt, sondern
      auch in einer neuen Excel-Datei namens 'Straftaten_Veraenderung.xlsx'
      gespeichert.
151
152     - **Anpassungen:**
153     - Falls die Sheet-Namen ein anderes Muster haben, musst du die Methode zur
      Extraktion des Jahres ('year = int(sheet.split('_')[-1])') entsprechend
      anpassen.
154
155     - 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
167     prozentuale Veränderung der Straftaten im Vergleich zum Vorjahr.
168 ---
169
170 Dieses Skript sollte dir dabei helfen, die gewünschten Analysen durchzuführen
171     . Passe es nach Bedarf an deine spezifischen Anforderungen und
172     Datenstrukturen an.

```

Listing 160: API Response Testfall 4 Prompt 3 Ausführung 3

```

1 Hier ist ein Python-Skript, das die beschriebenen Schritte ausführt. Das
2     Skript verwendet die 'pandas'-Bibliothek, um die Excel-Datei zu lesen und
3     die gewünschten Berechnungen durchzuführen.
4
5 ### Voraussetzungen
6 Stellen Sie sicher, dass Sie die 'pandas'-Bibliothek installiert haben. Falls
7     nicht, können Sie sie mit folgendem Befehl installieren:
8
9     '''bash
10     pip install pandas
11     '''
12
13 Zusätzlich benötigen Sie 'openpyxl' zum Lesen von Excel-Dateien im '.xlsx'-
14     Format:
15
16     '''bash
17     pip install openpyxl
18     '''
19
20 ### Python-Skript
21
22     '''python
23     import pandas as pd
24
25     # 1. Lesen Sie alle Sheets der Excel-Datei ein und speichern Sie jedes Sheet
26         in einem separaten DataFrame
27     excel_file = 'Fallzahlen.xlsx'
28     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
Muster 'Fallzahlen_Jahr'. Übersprungen.")
34         continue
35
36     # 2. Extrahiere den Wert der Spalte 'Straftaten_insgesamt' für die Zeile
'Berlin (PKS gesamt)'
37     # Überprüfen Sie, ob die Spaltennamen korrekt sind. Möglicherweise müssen
Sie 'Bezirke' anpassen.
38     # Hier wird angenommen, dass die Spalte mit Bezirken 'Bezirke' heißt
39     berlin_row = df[df['Bezirke'] == 'Berlin (PKS gesamt)']
40
41     if not berlin_row.empty:
42         straf_value = berlin_row['Straftaten_insgesamt'].values[0]
43         straftaten_dict[year] = straf_value
44     else:
45         print(f"'Berlin (PKS gesamt)' nicht in Sheet '{sheet_name}' gefunden.
")
46
47 # 3. Berechnen Sie die prozentuale Veränderung zum jeweiligen Vorjahr
48 # Sortieren der Jahre
49 sorted_years = sorted(straftaten_dict.keys())
50 percentage_changes = []
51
52 for i in range(1, len(sorted_years)):
53     current_year = sorted_years[i]
54     previous_year = sorted_years[i - 1]
55     current_value = straftaten_dict[current_year]
56     previous_value = straftaten_dict[previous_year]
57
58     # Prozentuale Veränderung berechnen
59     if previous_value != 0:
60         percent_change = ((current_value - previous_value) / previous_value)
* 100
61     else:
62         percent_change = None # Vermeiden von Division durch Null
63
64     percentage_changes.append({
65         'Jahr': current_year,
66         'Prozentuale Veränderung': percent_change
67     })
68
69 # 4. Speichern Sie die Ergebnisse in einem neuen DataFrame

```

```

70 result_df = pd.DataFrame(percentage_changes)
71
72 # Optional: Hinzufügen des ersten Jahres mit einer NaN für die Veränderung
73 result_df = pd.concat([
74     pd.DataFrame({'Jahr': [sorted_years[0]], 'Prozentuale Veränderung': [None
75         ]}),
76     result_df
77 ], ignore_index=True)
78
79 # Anzeige des Ergebnisses
80 print(result_df)
81
82 # Optional: Speichern des Ergebnisses in eine neue Excel-Datei
83 result_df.to_excel('Prozentuale_Veraenderung_Straftaten.xlsx', index=False)
84
85 ### Erklärung der Schritte
86
87 1. **Einlesen aller Sheets:**
88 - 'pd.read_excel' mit 'sheet_name=None' liest alle Sheets und speichert
89 sie in einem Dictionary, wobei die Schlüssel die Sheet-Namen sind.
90
91 2. **Extraktion der relevanten Daten:**
92 - Für jedes Sheet wird das Jahr aus dem Namen extrahiert.
93 - Es wird die Zeile gesucht, in der die Spalte 'Bezirke' den Wert 'Berlin
94   (PKS gesamt)' hat.
95 - Der Wert aus der Spalte 'Straftaten_insgesamt' wird extrahiert und im
96   Dictionary 'straftaten_dict' gespeichert.
97
98 3. **Berechnung der prozentualen Veränderung:**
99 - Die Jahre werden sortiert, um sicherzustellen, dass die Berechnungen in
100   chronologischer Reihenfolge erfolgen.
101 - Für jedes Jahr ab dem zweiten wird die prozentuale Veränderung im
102   Vergleich zum Vorjahr berechnet.
103
104 4. **Speichern der Ergebnisse:**
105 - Die Ergebnisse werden in einem neuen DataFrame 'result_df' gespeichert,
106   der die Jahre und die entsprechenden prozentualen Veränderungen enthält.
107 - Das Ergebnis wird angezeigt und optional in eine neue Excel-Datei '
108   Prozentuale_Veraenderung_Straftaten.xlsx' gespeichert.
109
110 ### Beispielausgabe
111
112 Angenommen, Ihre Daten umfassen die Jahre 2014 bis 2016 mit folgenden
    fiktiven Werten:

```

Jahr	Prozentuale Veränderung
2014	NaN
2015	5.0
2016	-3.8

```

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 161: API 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
35         gefunden.")
36         continue
37
38     # Extrahiere den Wert der Spalte 'Straftaten_insgesamt'
39     strftaten_insgesamt = berlin_data['Straftaten_insgesamt'].values[0]
40
41     daten_liste.append({'Jahr': jahr, 'Straftaten_insgesamt':
42     strftaten_insgesamt})
43
44 # 3. Erstellen Sie ein DataFrame aus den gesammelten Daten
45 daten_df = pd.DataFrame(daten_liste)
46
47 # Sortieren Sie das DataFrame nach Jahr aufsteigend
48 daten_df = daten_df.sort_values('Jahr').reset_index(drop=True)
49
50 # 4. Berechnen Sie die prozentuale Veränderung zum Vorjahr
51 daten_df['Prozentuale_Veraenderung'] = daten_df['Straftaten_insgesamt'].
52     pct_change() * 100
53
54 # Optional: Runden Sie die Prozentsätze auf zwei Dezimalstellen
55 daten_df['Prozentuale_Veraenderung'] = daten_df['Prozentuale_Veraenderung'].
56     round(2)
57
58 # 5. Erstellen Sie das Endergebnis DataFrame
59 ergebnis_df = daten_df[['Jahr', 'Prozentuale_Veraenderung']]
60
61 # Optional: Benennen Sie die Spalten auf Deutsch um
62 ergebnis_df.columns = ['Jahr', 'Prozentuale Veränderung (%)']
63
64 # 6. Anzeigen des Ergebnisses
65 print(ergebnis_df)
66
67 # 7. Speichern Sie das Ergebnis in einer neuen Excel-Datei (optional)
68 ergebnis_df.to_excel('Prozentuale_Veraenderung_Fallzahlen.xlsx', index=False)
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

```

Erläuterung der Schritte:

- **Einlesen aller Sheets:****
 - Das Skript verwendet 'pd.read_excel' mit 'sheet_name=None', um alle Sheets der Excel-Datei in ein Dictionary zu laden, wobei die Schlüssel die Sheet-Namen und die Werte die jeweiligen DataFrames sind.
- **Extrahieren der relevanten Daten:****
 - Für jedes Sheet wird das Jahr aus dem Sheet-Namen extrahiert.

```

74 - Es wird nach der Zeile gesucht, in der die Spalte 'Bezirke' den Wert '
    Berlin (PKS gesamt)' hat.
75 - Der Wert der Spalte 'Straftaten_insgesamt' wird extrahiert und in einer
    Liste gespeichert.
76
77 3. **Erstellen und Sortieren des DataFrames:**
78 - Die gesammelten Daten werden in ein neues DataFrame 'daten_df'
    umgewandelt.
79 - Das DataFrame wird nach dem Jahr aufsteigend sortiert, um
    sicherzustellen, dass die prozentuale Veränderung korrekt berechnet wird.
80
81 4. **Berechnung der prozentualen Veränderung:**
82 - Die Funktion 'pct_change()' von pandas berechnet die prozentuale Verä
    nderung zwischen den aufeinanderfolgenden Jahren.
83 - Das Ergebnis wird mit 100 multipliziert, um es in Prozent darzustellen.
84 - Optional werden die Prozentsätze auf zwei Dezimalstellen gerundet.
85
86 5. **Erstellen des Endergebnis-DataFrames:**
87 - Ein neues DataFrame 'ergebnis_df' wird erstellt, das nur die Spalten '
    Jahr' und 'Prozentuale_Veraenderung' enthält.
88 - Die Spalten werden optional auf Deutsch umbenannt.
89
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
    werden.
93
94 ### Beispielausgabe:
95
96 Angenommen, die Daten enthalten die Jahre von 2014 bis 2016 mit
    entsprechenden 'Straftaten_insgesamt' Werten. Die Ausgabe könnte wie folgt
    aussehen:
97
98 '''
99     Jahr    Prozentuale Veränderung (%)
100 0   2014                      NaN
101 1   2015                      5.23
102 2   2016                     -3.45
103 '''
104
105 **Hinweis:**
106 - Der erste Eintrag hat keinen Vorjahr-Wert, daher ist die prozentuale Verä
    nderung 'NaN'.
107 - 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.
108 - Passen Sie gegebenenfalls die Spaltennamen an, falls diese in der Excel-
    Datei anders lauten.
109
110 Falls Sie weitere Anpassungen benötigen oder Fragen haben, lassen Sie es mich
    wissen!

```

11.3.12 Komplette Antworten der API Requests Testfall 5

```

1 Hier ist ein Python-Skript, das die beschriebenen Anforderungen erfüllt. Das
  Skript verwendet die Bibliothek 'pandas', um die Excel-Datei zu lesen und
  die Daten zu verarbeiten. Stelle sicher, dass du 'pandas' und 'openpyxl'
  installiert hast, bevor du das Skript ausführst. Du kannst sie mit 'pip'
  installieren, falls sie noch nicht vorhanden sind:
2
3 '''bash
4 pip install pandas openpyxl
5 '''
6
7 ### Python-Skript
8
9 '''python
10 import pandas as pd
11
12 # Liste der Oberbezirke in Berlin
13 oberbezirke = [
14     'Mitte',
15     'Friedrichshain-Kreuzberg',
16     'Pankow',
17     'Charlottenburg-Wilmersdorf',
18     'Spandau',
19     'Steglitz-Zehlendorf',
20     'Tempelhof-Schöneberg',
21     'Neukölln',
22     'Treptow-Köpenick',
23     'Marzahn-Hellersdorf',
24     'Lichtenberg',
25     'Reinickendorf'
26 ]
27
28 def get_oberbezirk(bezeichnung):
29     """
30     Bestimmt den Oberbezirk basierend auf der Bezeichnung des Bezirks.
31     """
32     for oberbezirk in oberbezirke:
33         if bezeichnung.startswith(oberbezirk):
34             return oberbezirk
35     return None # Rückgabe von None, wenn kein Oberbezirk gefunden wird
36
37 def main():
38     # Pfad zur Excel-Datei
39     datei_pfad = 'Fallzahlen&HZ 2014-2023.xlsx'
40
41     try:
42         # Einlesen des spezifischen Sheets

```

```

43     df = pd.read_excel(datei_pfad, sheet_name='Fallzahlen_2023')
44 except FileNotFoundError:
45     print(f"Die Datei '{datei_pfad}' wurde nicht gefunden.")
46     return
47 except ValueError:
48     print("Das Sheet 'Fallzahlen_2023' existiert nicht in der Excel-Datei
49 .")
50     return
51
52 # Überprüfen, ob die notwendigen Spalten vorhanden sind
53 notwendige_spalten = ['Bezeichnung (Bezirksregion)', 'Raub']
54 for spalte in notwendige_spalten:
55     if spalte not in df.columns:
56         print(f"Die Spalte '{spalte}' ist in den Daten nicht vorhanden.")
57         return
58
59 # Hinzufügen einer neuen Spalte für den Oberbezirk
60 df['Oberbezirk'] = df['Bezeichnung (Bezirksregion)'].apply(get_oberbezirk)
61
62 # Entfernen von Einträgen, die keinem Oberbezirk zugeordnet werden können
63 df = df.dropna(subset=['Oberbezirk'])
64
65 # Gruppieren nach Oberbezirk und Finden des Unterbezirks mit den meisten
66 Raubdelikten
67 top_unterbezirke = df.loc[df.groupby('Oberbezirk')['Raub'].idxmax()][['
68 Oberbezirk', 'Bezeichnung (Bezirksregion)', 'Raub']]
69
70 # Ausgabe der Ergebnisse
71 print("Unterbezirk mit den meisten Raubdelikten pro Oberbezirk (2023):\n"
72 )
73 for index, row in top_unterbezirke.iterrows():
74     oberbezirk = row['Oberbezirk']
75     unterbezirk = row['Bezeichnung (Bezirksregion)']
76     raub = row['Raub']
77     print(f"Oberbezirk: {oberbezirk}")
78     print(f"    Unterbezirk: {unterbezirk}")
79     print(f"    Anzahl Raubdelikte: {raub}\n")
80
81 if __name__ == "__main__":
82     main()
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```



```

87 3. **Funktion 'get_oberbezirk'**: Diese Funktion nimmt die Bezeichnung eines
    Bezirks und bestimmt, zu welchem Oberbezirk er gehört, indem sie prüft, ob
    die Bezeichnung mit einem der Oberbezirke beginnt.
88
89 4. **Hauptfunktion 'main'**:
90     - **Einlesen der Excel-Datei**: Das Skript versucht, das spezifische
    Sheet 'Fallzahlen_2023' aus der Excel-Datei zu lesen. Fehler beim Einlesen
    werden abgefangen und gemeldet.
91     - **Überprüfen der notwendigen Spalten**: Es wird geprüft, ob die benö
    tigten Spalten vorhanden sind.
92     - **Zuweisen der Oberbezirke**: Eine neue Spalte 'Oberbezirk' wird
    hinzugefügt, die den zugehörigen Oberbezirk für jeden Unterbezirk enthält.
93     - **Filtern nicht zuzuordenbarer Einträge**: Einträge, die keinem
    Oberbezirk zugeordnet werden können ('None'), werden entfernt.
94     - **Gruppieren und Finden der Top-Unterbezirke**: Das Skript gruppiert
    die Daten nach Oberbezirk und findet innerhalb jeder Gruppe den
    Unterbezirk mit der höchsten Anzahl an Raubdelikten.
95     - **Ausgabe der Ergebnisse**: Für jeden Oberbezirk wird der entsprechende
    Top-Unterbezirk samt der Anzahl der Raubdelikte ausgegeben.
96
97 ### Ausführung des Skripts
98
99 1. **Speichern des Skripts**: Speichere das obige Skript in einer Datei, zum
    Beispiel 'top_raub_unterbezirke.py'.
100
101 2. **Platzierung der Excel-Datei**: Stelle sicher, dass sich die Excel-Datei
    'Fallzahlen&HZ 2014-2023.xlsx' im selben Verzeichnis wie das Skript
    befindet oder passe den 'datei_pfad' entsprechend an.
102
103 3. **Ausführen des Skripts**:
104     '''bash
105     python top_raub_unterbezirke.py
106     '''
107
108 Nach der Ausführung erhältst du eine Ausgabe, die für jeden Oberbezirk den
    Unterbezirk mit den meisten Raubdelikten im Jahr 2023 anzeigt. Die Ausgabe
    könnte beispielsweise wie folgt aussehen:
109
110 '''
111 Unterbezirk mit den meisten Raubdelikten pro Oberbezirk (2023):
112
113 Oberbezirk: Mitte
114     Unterbezirk: Alexanderplatz
115     Anzahl Raubdelikte: 199
116
117 Oberbezirk: Pankow
118     Unterbezirk: ...
119     Anzahl Raubdelikte: ...
120
121 ...
122 '''
123

```

124 Dieses Skript kann weiter angepasst werden, um zusätzliche Analysen durchzuführen oder die Ergebnisse in eine Datei zu exportieren, falls gewünscht.

Listing 163: API Response Testfall 5 Prompt 1 Ausführung 1

```
1 Natürlich! Hier ist ein Python-Skript, das die beschriebenen Anforderungen
  erfüllt. Es verwendet die 'pandas'-Bibliothek, um die Excel-Datei zu lesen
  und die Daten zu verarbeiten. Das Skript geht davon aus, dass die ersten
  zwei Stellen des 'LOR-Schlüssel (Bezirksregion)' die Zugehörigkeit zu
  einem 'Oberbezirk' bestimmen. Falls diese Annahme nicht korrekt ist, musst
  du den Code entsprechend anpassen.
2
3 ### Voraussetzungen
4
5 1. **Python Installation**: Stelle sicher, dass Python auf deinem System
  installiert ist. Du kannst die neueste Version von [python.org](https://
  www.python.org/downloads/) herunterladen.
6
7 2. **Benötigte Bibliotheken installieren**: Installiere die erforderlichen
  Bibliotheken mit 'pip', falls sie noch nicht installiert sind:
8
9 ```bash
10 pip install pandas openpyxl
11 ```
12
13 ### Python-Skript
14
15 ```python
16 import pandas as pd
17
18 # Pfad zur Excel-Datei
19 excel_file = 'Fallzahlen&HZ 2014-2023.xlsx'
20
21 # Name des Sheets
22 sheet_name = 'Fallzahlen_2023'
23
24 # Laden des Sheets in einen DataFrame
25 try:
26     df = pd.read_excel(excel_file, sheet_name=sheet_name)
27 except FileNotFoundError:
28     print(f"Die Datei '{excel_file}' wurde nicht gefunden.")
29     exit(1)
30 except ValueError:
31     print(f"Das Sheet '{sheet_name}' existiert nicht in der Datei.")
32     exit(1)
33
34 # Überprüfen der relevanten Spalten
35 required_columns = [
36     'LOR-Schlüssel (Bezirksregion)',
37     'Bezeichnung (Bezirksregion)',
38     'Raub'
39 ]
40
```

```

41 for col in required_columns:
42     if col not in df.columns:
43         print(f"Die erforderliche Spalte '{col}' ist im Sheet nicht vorhanden
44             .")
45         exit(1)
46 # Entfernen von Zeilen, die nicht zugeordnet sind (optional)
47 # Angenommen, LOR-Schlüssel '129900', '999900', '999999' sind nicht
48     zuzuordnen
49 invalid_keys = ['129900', '999900', '999999']
50 df = df[~df['LOR-Schlüssel (Bezirksregion)'].astype(str).str.startswith(tuple
51     (invalid_keys))]
52 # Extrahieren des Oberbezirkscodes (angenommen, die ersten 2 Stellen)
53 df['Oberbezirk_Code'] = df['LOR-Schlüssel (Bezirksregion)'].astype(str).str
54     [:2]
55 # Optional: Mapping der Oberbezirk-Codes zu Namen
56 # Dies muss entsprechend deiner Daten angepasst werden
57 oberbezirk_mapping = {
58     '10': 'Mitte',
59     '11': 'Pankow',
60     '12': 'Charlottenburg-Wilmersdorf',
61     '13': 'Friedrichshain-Kreuzberg',
62     '14': 'Lichtenberg',
63     '15': 'Marzahn-Hellersdorf',
64     '16': 'Treptow-Köpenick',
65     '17': 'Tempelhof-Schöneberg',
66     '18': 'Neukölln',
67     '19': 'Steglitz-Zehlendorf',
68     # Füge weitere Codes und Namen hinzu, falls vorhanden
69 }
70 df['Oberbezirk'] = df['Oberbezirk_Code'].map(oberbezirk_mapping)
71 # Überprüfen auf fehlende Zuordnungen
72 missing_mapping = df[df['Oberbezirk'].isna()]['Oberbezirk_Code'].unique()
73 if len(missing_mapping) > 0:
74     print("Warnung: Einige Oberbezirk-Codes konnten nicht zugeordnet werden:"
75         , missing_mapping)
76     # Optional: Weiterverarbeiten oder Abbruch
77     # exit(1)
78 # Gruppieren nach Oberbezirk und Finden des Unterbezirks mit den meisten
79     Raubdelikten
80 result = df.loc[df.groupby('Oberbezirk')['Raub'].idxmax()][['Oberbezirk', '
81     Bezeichnung (Bezirksregion)', 'Raub']]
82 # Sortieren nach Oberbezirk
83 result = result.sort_values('Oberbezirk')
84 # Ergebnis anzeigen
85

```

```

86 print("Unterbezirk mit den meisten Raubdelikten pro Oberbezirk:")
87 print(result.to_string(index=False))
88
89 # Optional: Ergebnis in eine neue Excel-Datei speichern
90 output_file = 'Meiste_Raubdelikte_pro_Oberbezirk.xlsx'
91 result.to_excel(output_file, index=False)
92 print(f"\nDas Ergebnis wurde in '{output_file}' gespeichert.")
93 '''
94
95 ### Erklärung des Skripts
96
97 1. **Importieren der Bibliotheken**: 'pandas' wird verwendet, um die Excel-
    Datei zu lesen und zu verarbeiten.
98
99 2. **Einlesen der Excel-Datei**: Das Skript versucht, die angegebene Excel-
    Datei und das Sheet zu laden. Bei Fehlern wird eine entsprechende
    Nachricht ausgegeben und das Skript beendet.
100
101 3. **Überprüfen der erforderlichen Spalten**: Stellt sicher, dass die
    notwendigen Spalten vorhanden sind.
102
103 4. **Entfernen von nicht zugeordneten Bezirken**: Optional wird eine
    Filterung vorgenommen, um Bezirke auszuschließen, die nicht zugeordnet
    werden können.
104
105 5. **Extrahieren des Oberbezirks-Codes**: Die ersten zwei Stellen des 'LOR-
    Schlüssel (Bezirksregion)' werden als 'Oberbezirk_Code' extrahiert.
106
107 6. **Mapping der Oberbezirk-Codes zu Namen**: Ein Wörterbuch ('
    oberbezirk_mapping') ordnet die Codes den entsprechenden Namen zu. Dieses
    Dictionary sollte entsprechend deiner tatsächlichen Daten angepasst werden
    .
108
109 7. **Gruppierung und Finden des maximalen Werts**: Für jeden 'Oberbezirk'
    wird der 'Unterbezirk' mit den meisten Raubdelikten identifiziert.
110
111 8. **Ausgabe und Speicherung des Ergebnisses**: Die Ergebnisse werden in der
    Konsole angezeigt und optional in eine neue Excel-Datei gespeichert.
112
113 ### Anpassungen
114
115 - **Oberbezirk Mapping**: Stelle sicher, dass das 'oberbezirk_mapping'-
    Dictionary alle relevanten Oberbezirks-Codes und deren Namen enthält. Füge
    weitere Einträge hinzu, falls erforderlich.
116
117 - **Filterung nicht zugeordneter Bezirke**: Passe die 'invalid_keys'
    entsprechend deinen Daten an oder entferne diese Zeilen, wenn sie relevant
    sind.
118
119 - **Spaltennamen überprüfen**: Stelle sicher, dass die Spaltennamen exakt mit
    denen in deiner Excel-Datei übereinstimmen. Passe sie andernfalls im
    Skript an.

```

```

120
121 ### Ausführen des Skripts
122
123 1. Speichere das Skript in einer Datei, z.B. 'raubdelikte_analyse.py'.
124
125 2. Stelle sicher, dass sich die Excel-Datei 'Fallzahlen&HZ 2014-2023.xlsx' im
    gleichen Verzeichnis befindet oder passe den Pfad im Skript entsprechend
    an.
126
127 3. Führe das Skript über die Kommandozeile aus:
128
129 '''bash
130 python raubdelikte_analyse.py
131 '''
132
133 Nach der Ausführung erhältst du eine Ausgabe, die für jeden Oberbezirk den
    Unterbezirk mit den meisten Raubdelikten zeigt, sowie eine Excel-Datei mit
    den Ergebnissen.
134
135 ### Beispielausgabe
136
137 '''
138 Unterbezirk mit den meisten Raubdelikten pro Oberbezirk:
139
140     Oberbezirk           Bezeichnung (Bezirksregion)   Raub
141     Mitte               Mitte               884
142     Pankow              Tiergarten Süd           47
143     Charlottenburg-Wilmersdorf Regierungsviertel       44
144     Friedrichshain-Kreuzberg      Alexanderplatz       199
145     ...                   ...                   ...
146
147 Das Ergebnis wurde in 'Meiste_Raubdelikte_pro_Oberbezirk.xlsx' gespeichert.
148 '''
149
150 Dieses Skript sollte dir einen guten Startpunkt bieten. Passe es nach Bedarf
    an deine spezifischen Daten und Anforderungen an!

```

Listing 164: API Response Testfall 5 Prompt 1 Ausführung 2

```

1 Natürlich! Hier ist ein Python-Skript, das die gewünschten Anforderungen erfü
  llt. Dieses Skript verwendet die Bibliothek 'pandas', um die Excel-Datei
  zu lesen und die Daten zu verarbeiten. Es wird davon ausgegangen, dass der
  **LOR-Schlüssel** (Bezirksregion-Identifizier) die Hierarchie zwischen
  Oberbezirk und Unterbezirk bestimmt. In diesem Beispiel nehme ich an, dass
  die ersten zwei Stellen des **LOR-Schlüssels** den Oberbezirk
  identifizieren und die restlichen Stellen den Unterbezirk.
2
3 ### Schritt-für-Schritt-Anleitung:
4
5 1. **Installation der benötigten Bibliotheken**:
6     Stellen Sie sicher, dass Sie die Bibliothek 'pandas' und 'openpyxl'
     installiert haben. Falls nicht, können Sie diese 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     file_path = 'Fallzahlen&HZ 2014-2023.xlsx'
19     sheet_name = 'Fallzahlen_2023'
20
21     # Excel-Datei einlesen
22     df = pd.read_excel(file_path, sheet_name=sheet_name)
23
24     # Anzeige der ersten Zeilen zur Überprüfung (optional)
25     # print(df.head())
26
27     # Annahme: Der LOR-Schlüssel hat mindestens 5 Stellen
28     # Extrahieren des Oberbezirksskennung (erste zwei Stellen)
29     df['Oberbezirk_Key'] = df['LOR-Schlüssel'].astype(str).str[:2]
30
31     # Extrahieren des Unterbezirksskennung (ganzer Schlüssel)
32     df['Unterbezirk_Key'] = df['LOR-Schlüssel'].astype(str)
33
34     # Gruppieren nach Oberbezirk_Key und Bezeichnung des Oberbezirks
35     # Hier nehmen wir an, dass die Bezeichnung des Oberbezirks dieselbe ist für
36     # alle Unterbezirke
37     oberbezirke = df.groupby('Oberbezirk_Key')['Bezeichnung'].first().
38     reset_index()
39     oberbezirke = oberbezirke.rename(columns={'Bezeichnung': 'Oberbezirk_Name'
40     })
41
42     # Merge der Oberbezirk Informationen zurück zum Haupt-DataFrame
43     df = df.merge(oberbezirke, on='Oberbezirk_Key', how='left')
44
45     # Gruppieren nach Oberbezirk und Finden des Unterbezirks mit den meisten
46     # Raubdelikten
47     # Annahme: Die Spalte für Raubdelikte heißt 'Raub'
48     result = df.groupby('Oberbezirk_Name').apply(
49         lambda x: x.loc[x['Raub'].idxmax()][['Bezeichnung', 'Raub']]
50     ).reset_index()
51
52     # Umbenennen der Spalten für Klarheit
53     result = result.rename(columns={
54         'Bezeichnung': 'Unterbezirk_mit_meisten_Raub',
55         'Raub': 'Anzahl_Raub'
56     })
57
58     # Anzeige des Ergebnisses
59     print(result)
60

```

```

57 # Optional: Speichern des Ergebnisses in eine CSV-Datei
58 result.to_csv('Unterbezirk_mit_meisten_Raub_pro_Oberbezirk_2023.csv',
59               index=False)
60
61 ### Erklärung des Skripts:
62
63 1. **Daten einlesen**:
64    - Das Skript liest das angegebene Excel-Sheet 'Fallzahlen_2023' ein.
65
66 2. **Extrahieren von Oberbezirk und Unterbezirk**:
67    - Es wird angenommen, dass die ersten zwei Stellen des 'LOR-Schlüssel'
68      den Oberbezirk identifizieren. Diese werden in der neuen Spalte '
69      Oberbezirk_Key' gespeichert.
70    - Die gesamte 'LOR-Schlüssel' wird als 'Unterbezirk_Key' beibehalten.
71
72 3. **Gruppierung und Bestimmung des Oberbezirksnamens**:
73    - Die Bezeichnungen der Oberbezirke werden extrahiert, indem der erste
74      Eintrag für jeden Oberbezirk genommen wird.
75
76 4. **Bestimmung des Unterbezirks mit den meisten Raubdelikten**:
77    - Für jeden Oberbezirk wird der Unterbezirk mit der höchsten Anzahl an
78      Raubdelikten ermittelt.
79
80 5. **Ergebnis anzeigen und speichern**:
81    - Das Ergebnis wird sowohl im Terminal angezeigt als auch optional in
82      einer CSV-Datei gespeichert.
83
84 ### Beispielausgabe:
85
86 '''
87
88     Oberbezirk_Name Unterbezirk_mit_meisten_Raub  Anzahl_Raub
89 0             Mitte                Mitte           884
90 1   Tiergarten Süd          Tiergarten Süd           47
91 2   Regierungsviertel      Regierungsviertel           44
92 3   Alexanderplatz         Alexanderplatz          199
93 4   Brunnenstraße Süd     Brunnenstraße Süd            51
94 ...                ...                ...
95 '''
96
97 Diese Ausgabe zeigt für jeden Oberbezirk den Unterbezirk mit den meisten
98 Raubdelikten sowie die entsprechende Anzahl dieser Delikte.
99
100 ### Hinweise:
101
102 - **Anpassung der Schlüssel**: Stellen Sie sicher, dass die Annahme über den
103   Aufbau des 'LOR-Schlüssel' korrekt ist. Falls der **Oberbezirk** anders
104   kodiert ist, passen Sie die Extraktion entsprechend an.
105
106 - **Spaltennamen überprüfen**: Vergewissern Sie sich, dass die Spaltennamen
107   ('LOR-Schlüssel', 'Bezeichnung', 'Raub' usw.) genau mit denen in
108   Ihrer Excel-Datei übereinstimmen. Andernfalls passen Sie die Namen im
109   Skript an.

```

```

97 - **Fehlerbehandlung**: Für ein robustes Skript sollten zusätzliche
    Fehlerbehandlungen hinzugefügt werden, zum Beispiel um fehlende Werte oder
    unerwartete Datenformate zu behandeln.
98
99 Ich hoffe, dieses Skript hilft Ihnen weiter! Wenn Sie weitere Anpassungen ben
    ötigen oder Fragen haben, stehe ich gerne zur Verfügung.

```

Listing 165: API Response Testfall 5 Prompt 1 Ausführung 3

```

1 Hier ist ein Python-Skript, das die gewünschten Anforderungen erfüllt. Das
  Skript verwendet die Bibliothek 'pandas', um die Excel-Datei zu lesen und
  die Daten zu verarbeiten.
2
3 ## Annahmen und Vorgehensweise
4
5 1. **Oberbezirk Identifikation**:
6   - **Oberbezirk** wird angenommen als Einträge mit einem 'LOR-Schlüssel',
     der 5-stellig ist und mit '000' endet (z.B. '10000' für Mitte).
7   - **Unterbezirk** sind Einträge, die nicht diese Kriterien erfüllen und
     deren 'LOR-Schlüssel' mit dem gleichen Anfang wie der Oberbezirk beginnen.
8
9 2. **Zuordnung der Unterbezirke**:
10  - Jedes Unterbezirk wird dem Oberbezirk zugeordnet, dessen 'LOR-Schlüssel'
    der längste Präfix des Unterbezirks-Schlüssels ist. Dies ermöglicht eine
    flexible Zuordnung, selbst wenn die Hierarchie tiefer ist.
11
12 3. **Ermittlung der Unterbezirke mit den meisten Raubdelikten**:
13  - Für jeden Oberbezirk wird der Unterbezirk mit der höchsten Anzahl an
    Raubdelikten ('Raub') ermittelt.
14
15 ## Benötigte Bibliotheken
16
17 Stellen Sie sicher, dass die folgenden Bibliotheken installiert sind. Falls
    nicht, können Sie sie mit 'pip' installieren:
18
19 '''bash
20 pip install pandas openpyxl
21 '''
22
23 ## Python-Skript
24
25 '''python
26 import pandas as pd
27
28 def finde_oberbezirke_mit_meisten_raub(excel_datei, sheet_name):
29     # Excel-Datei lesen
30     try:
31         df = pd.read_excel(excel_datei, sheet_name=sheet_name, dtype={'LOR-
    Schlüssel': str})
32     except Exception as e:
33         print(f"Fehler beim Lesen der Excel-Datei: {e}")
34         return
35

```



```

36     # Sicherstellen, dass 'LOR-Schlüssel' als String gelesen werden und fü
hrende Nullen beibehalten
37     df['LOR-Schlüssel'] = df['LOR-Schlüssel'].str.strip().str.zfill(6)
38
39     # Entfernen von nicht zuzuordenenden Bezirken
40     df = df[~df['Bezeichnung (Bezirksregion)'].str.contains('nicht zuzuordnen
', na=False)]
41
42     # Identifizieren der Oberbezirke: 5-stelliger LOR-Schlüssel, der mit
'000' endet
43     oberbezke = df[df['LOR-Schlüssel'].str.endswith('000') & (df['LOR-Schlü
ssel'].str.len() == 5)]
44
45     if oberbezke.empty:
46         print("Keine Oberbezirke gefunden. Überprüfen Sie die Kriterien zur
Identifikation der Oberbezirke.")
47         return
48
49     # Liste der Oberbezirke
50     oberbezke_list = oberbezke[['LOR-Schlüssel', 'Bezeichnung (Bezirksregion)
']].to_dict('records')
51
52     # Ergebnisse speichern
53     ergebnisse = []
54
55     for ober in oberbezke_list:
56         ober_code = ober['LOR-Schlüssel']
57         ober_name = ober['Bezeichnung (Bezirksregion)']
58
59         # Finden der Unterbezirke, die mit dem Oberbezirkscode beginnen
60         # Annahme: Der Oberbezirkscode ohne die letzten drei Nullen ist der
Präfix
61         prefix = ober_code[:-3] # z.B. '10' aus '10000'
62
63         # Unterbezirke finden
64         unterbezke = df[
65             (df['LOR-Schlüssel'].str.startswith(prefix)) &
66             (df['LOR-Schlüssel'] != ober_code)
67         ]
68
69         if unterbezke.empty:
70             print(f"Keine Unterbezirke für Oberbezirk {ober_name} gefunden.")
71             continue
72
73         # Unterbezirk mit den meisten Raubdelikten finden
74         max_raub_idx = unterbezke['Raub'].idxmax()
75         max_raub = unterbezke.loc[max_raub_idx]
76         unterbez_name = max_raub['Bezeichnung (Bezirksregion)']
77         raub_count = max_raub['Raub']
78
79         ergebnisse.append({
80             'Oberbezirk': ober_name,

```

```

81         'Unterbezirk': unterbez_name,
82         'Raubdelikte': raub_count
83     })
84
85     # Ergebnisse anzeigen
86     print("Unterbezirke mit den meisten Raubdelikten pro Oberbezirk:")
87     for res in ergebnisse:
88         print(f"Oberbezirk: {res['Oberbezirk']}, "
89               f"Unterbezirk: {res['Unterbezirk']}, "
90               f"Raubdelikte: {res['Raubdelikte']}")
91
92 if __name__ == "__main__":
93     excel_datei = 'Fallzahlen&HZ 2014-2023.xlsx' # Pfad zur Excel-Datei
94     sheet_name = 'Fallzahlen_2023' # Name des Sheets
95     finde_oberbezirke_mit_meisten_raub(excel_datei, sheet_name)
96 '''
97
98 ## Erklärung des Skripts
99
100 1. **Lesen der Excel-Datei**:
101     - Das Skript liest das angegebene Sheet der Excel-Datei ein.
102     - Der 'LOR-Schlüssel' wird als String eingelesen, um führende Nullen zu
103       bewahren.
104
105 2. **Filtern der Oberbezirke**:
106     - Oberbezirke werden anhand des 'LOR-Schlüssel' identifiziert, der 5-
107       stellig ist und mit '000' endet.
108
109 3. **Zuordnung der Unterbezirke**:
110     - Für jeden Oberbezirk wird ein Präfix aus dem 'LOR-Schlüssel' extrahiert
111       (die ersten zwei Stellen).
112     - Unterbezirke, die mit diesem Präfix beginnen (aber nicht dem Oberbezirk
113       selbst entsprechen), werden zugeordnet.
114
115 4. **Ermittlung des Unterbezirks mit den meisten Raubdelikten**:
116     - Innerhalb der zugeordneten Unterbezirke wird der mit dem höchsten Wert
117       in der Spalte 'Raub' ausgewählt.
118
119 5. **Ausgabe der Ergebnisse**:
120     - Die Ergebnisse werden in der Konsole ausgegeben, wobei jeder Oberbezirk
121       und der entsprechende Unterbezirk mit den meisten Raubdelikten angezeigt
122       werden.
123
124 ## Hinweise
125
126 - **Datensäuberung**: Stellen Sie sicher, dass die Spaltennamen in Ihrer
127   Excel-Datei exakt mit denen im Skript übereinstimmen. Eventuell müssen Sie
128   die Spaltennamen anpassen, falls sie abweichen.
129
130 - **Fehlerbehandlung**: Das Skript enthält grundlegende Fehlerbehandlungen,
131   um Probleme beim Lesen der Datei oder beim Fehlen von Oberbezirken zu
132   melden.
133
134 - **Anpassungen**: Je nach tatsächlicher Struktur der 'LOR-Schlüssel' und der

```

```

    administrativen Hierarchie in Ihrer Datenquelle könnten Anpassungen
    erforderlich sein.
122
123 ## Beispielausgabe
124
125 '''
126 Unterbezirke mit den meisten Raubdelikten pro Oberbezirk:
127 Oberbezirk: Mitte, Unterbezirk: Alexanderplatz, Raubdelikte: 199
128 Oberbezirk: ...
129 ...
130 '''
131
132 Dieses Skript sollte Ihnen eine gute Grundlage bieten, um die gewünschten
    Analysen durchzuführen. Passen Sie es entsprechend den genauen
    Anforderungen und der Struktur Ihrer Daten an.

```

Listing 166: API Response Testfall 5 Prompt 1 Ausführung 4

```

1 Natürlich! Hier ist ein Python-Skript, das die gewünschten Anforderungen erfü
    llt. Dieses Skript verwendet die 'pandas' Bibliothek, um die Excel-Datei
    zu lesen und die Analyse durchzuführen. Stellen Sie sicher, dass Sie die
    notwendigen Bibliotheken installiert haben, indem Sie 'pip install pandas
    openpyxl' ausführen, falls Sie dies noch nicht getan haben.
2
3 ## Annahmen und Vorgehensweise
4
5 1. **Oberbezirk Definition:** Da die genaue Struktur der Ober- und
    Unterbezirke aus den bereitgestellten Daten nicht eindeutig ersichtlich
    ist, gehe ich davon aus, dass die ersten drei Stellen des 'LOR-Schlüssel'
    den Oberbezirk identifizieren. Diese Annahme basiert auf typischen
    Kodierungssystemen, kann jedoch je nach spezifischer Verwendung variieren.
    Passen Sie dies bei Bedarf an.
6
7 2. **Daten Bereinigung:** Das Skript schließt spezielle Einträge wie "nicht
    zuzuordnen" und Gesamtdatensätze ("Berlin (PKS gesamt)") aus der Analyse
    aus, da sie keine spezifischen Unterbezirke repräsentieren.
8
9 3. **Maximale Raubdelikte:** Für jeden Oberbezirk wird der Unterbezirk mit
    den meisten Raubdelikten identifiziert und ausgegeben.
10
11 ## Python-Skript
12
13 '''python
14 import pandas as pd
15
16 # Pfad zur Excel-Datei
17 excel_datei = 'Fallzahlen&HZ 2014-2023.xlsx'
18
19 # Lesen des Sheets 'Fallzahlen_2023'
20 df = pd.read_excel(excel_datei, sheet_name='Fallzahlen_2023')
21
22 # Anzeigen der ersten Zeilen zur Überprüfung (optional)
23 # print(df.head())

```

```

24
25 # Sicherstellen, dass 'LOR-Schlüssel' als String behandelt wird
26 df['LOR-Schlüssel'] = df['LOR-Schlüssel'].astype(str)
27
28 # Extrahieren der Oberbezirk-Information aus den ersten 3 Zeichen des LOR-
    Schlüssels
29 # Passen Sie dies an, falls die Struktur anders ist
30 df['Oberbezirk_Key'] = df['LOR-Schlüssel'].str[:3]
31
32 # Optional: Definieren Sie eine Mapping-Tabelle für Oberbezirk_Key zu
    Oberbezirk_Name
33 # Hier ist ein Beispiel. Sie müssen dies basierend auf Ihrer spezifischen
    Kodierung anpassen.
34 oberbezirk_mapping = {
35     '100': 'Mitte',
36     '110': 'Cluster A', # Beispielnamen
37     '120': 'Cluster B',
38     # Fügen Sie weitere Zuordnungen hier hinzu
39     # ...
40 }
41
42 # Hinzufügen einer Spalte für den Oberbezirk-Namen
43 df['Oberbezirk'] = df['Oberbezirk_Key'].map(oberbezirk_mapping)
44
45 # Falls bestimmte Keys nicht im Mapping vorhanden sind, können Sie den Key
    als Namen verwenden
46 df['Oberbezirk'] = df['Oberbezirk'].fillna(df['Oberbezirk_Key'])
47
48 # Bereinigen der Daten: Entfernen von Einträgen, die nicht zugeordnet werden
    können
49 ausgeschlossene_terms = ['nicht zuzuordnen', 'gesamt']
50
51 df_clean = df[~df['Bezeichnung (Bezirksregion)'].str.contains('|'.join(
    ausgeschlossene_terms), na=False)]
52
53 # Gruppieren nach Oberbezirk und Finden des Unterbezirks mit den meisten
    Raubdelikten
54 result = df_clean.loc[df_clean.groupby('Oberbezirk')['Raub'].idxmax()]['
    Oberbezirk', 'Bezeichnung (Bezirksregion)', 'Raub']
55
56 # Sortieren der Ergebnisse nach Oberbezirk (optional)
57 result = result.sort_values('Oberbezirk')
58
59 # Ausgabe der Ergebnisse
60 for index, row in result.iterrows():
61     print(f"Oberbezirk: {row['Oberbezirk']}, Unterbezirk mit den meisten
        Raubdelikten: {row['Bezeichnung (Bezirksregion)']} ({row['Raub']}
        Raubdelikte)")
62
63 # Optional: Speichern der Ergebnisse in einer neuen Excel- oder CSV-Datei
64 # result.to_excel('Ergebnis_Most_Raubdelikte.xlsx', index=False)
65 # oder

```

```

66 # result.to_csv('Ergebnis_Most_Raubdelikte.csv', index=False)
67 '''
68
69 ## Erklärung der Schritte
70
71 1. **Importieren der Bibliotheken:**
72 - 'pandas' wird für die Datenverarbeitung verwendet.
73
74 2. **Lesen der Excel-Datei:**
75 - Das Skript liest das spezifische Sheet 'Fallzahlen_2023' aus der Excel
76 -Datei.
77
78 3. **Vorverarbeitung:**
79 - Der 'LOR-Schlüssel' wird als String behandelt, und die ersten drei
80 Zeichen werden extrahiert, um den 'Oberbezirk' zu identifizieren.
81 - Eine Mapping-Tabelle ('oberbezirk_mapping') wird verwendet, um die Schlü
82 ssel den tatsächlichen Oberbezirksnamen zuzuordnen. Sie müssen diese
83 Tabelle entsprechend Ihrer tatsächlichen Datenstruktur anpassen.
84
85 4. **Datenbereinigung:**
86 - Einträge, die "nicht zuzuordnen" oder "gesamt" in der 'Bezeichnung (
87 Bezirksregion)' enthalten, werden ausgeschlossen, da sie keine
88 spezifischen Unterbezirke darstellen.
89
90 5. **Gruppierung und Identifikation:**
91 - Die Daten werden nach 'Oberbezirk' gruppiert, und innerhalb jeder Gruppe
92 wird der Unterbezirk mit den höchsten Raubdelikten identifiziert.
93
94 6. **Ausgabe:**
95 - Die Ergebnisse werden in der Konsole ausgegeben. Optional können Sie die
96 Ergebnisse auch in eine neue Excel- oder CSV-Datei speichern, indem Sie
97 die auskommentierten Zeilen am Ende des Skripts aktivieren.
98
99 ## Hinweise
100
101 - **Mapping-Anpassung:** Die 'oberbezirk_mapping'-Diktionary muss basierend
auf Ihrer spezifischen 'LOR-Schlüssel'-Struktur und den tatsächlichen
Oberbezirksnamen angepasst werden. Wenn die ersten drei Stellen des 'LOR-
Schlüssels' nicht den Oberbezirken entsprechen, passen Sie die Extraktion
entsprechend an.
- **Bibliotheken installieren:** Falls Sie 'pandas' oder 'openpyxl' noch
nicht installiert haben, können Sie dies mit den folgenden Befehlen tun:
'''bash
pip install pandas openpyxl
'''
- **Fehlerbehebung:** Stellen Sie sicher, dass die Spaltennamen im Skript
exakt mit denen in Ihrer Excel-Datei übereinstimmen. Falls es Abweichungen
gibt, passen Sie die Spaltennamen im Skript entsprechend an.

```

```

102 - **Erweiterungen:** Sie können das Skript erweitern, um weitere Analysen
    durchzuführen oder die Ergebnisse visuell darzustellen, z.B. mit
    Diagrammen.
103
104 Ich hoffe, dieses Skript hilft Ihnen bei Ihrer Analyse! Bei weiteren Fragen
    stehe ich gerne zur Verfügung.

```

Listing 167: API Response Testfall 5 Prompt 1 Ausführung 5

```

1 Gerne helfe ich Ihnen dabei, ein Python-Skript zu erstellen, das die gewü
  nschten Informationen aus Ihrer Excel-Datei extrahiert. Im Folgenden
  finden Sie ein vollständiges Skript, das die Unterbezirke mit den meisten
  Raubdelikten für jeden Oberbezirk aus dem Sheet 'Fallzahlen_2023'
  ausliest und ausgibt.
2
3 ### Voraussetzungen
4
5 Stellen Sie sicher, dass Sie die notwendigen Python-Pakete installiert haben.
  Sie benötigen vor allem 'pandas' und 'openpyxl'. Sie können diese Pakete
  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&HZ 2014-2023.xlsx'
18
19 # Name des Sheets
20 sheet_name = 'Fallzahlen_2023'
21
22 # Oberbezirke-Liste
23 oberbezirke = [
24     'Mitte',
25     'Friedrichshain-Kreuzberg',
26     'Pankow',
27     'Charlottenburg-Wilmersdorf',
28     'Spandau',
29     'Steglitz-Zehlendorf',
30     'Tempelhof-Schöneberg',
31     'Neukölln',
32     'Treptow-Köpenick',
33     'Marzahn-Hellersdorf',
34     'Lichtenberg',
35     'Reinickendorf'
36 ]
37
38 # Daten aus dem Excel-Sheet auslesen

```

```

39 df = pd.read_excel(excel_datei, sheet_name=sheet_name)
40
41 # Letzte zwei Zeilen (Gesamtwerte) entfernen
42 df = df.iloc[:-2].reset_index(drop=True)
43
44 # Spaltennamen anpassen, falls nötig (Beispiel: Leerzeichen entfernen)
45 df.columns = [col.strip() for col in df.columns]
46
47 # Liste zur Speicherung der Ergebnisse
48 ergebnisse = []
49
50 # Variable zur Speicherung des aktuellen Oberbezirks
51 aktueller_oberbezirk = None
52
53 # Dictionary zur Speicherung der maximalen Raubdelikte pro Oberbezirk
54 max_raub_dict = {}
55
56 for index, row in df.iterrows():
57     bezirk = row['Bezeichnung (Bezirksregion)']
58
59     if bezirk in oberbezirke:
60         # Neuer Oberbezirk gefunden
61         aktueller_oberbezirk = bezirk
62     else:
63         if aktueller_oberbezirk is not None:
64             # Unterbezirk des aktuellen Oberbezirks
65             unterbezirk = bezirk
66             raub = row['Raub']
67
68             # Überprüfen, ob dieser Unterbezirk mehr Raubdelikte hat als
69             # vorherige
70             if aktueller_oberbezirk not in max_raub_dict:
71                 max_raub_dict[aktueller_oberbezirk] = {'Unterbezirk':
72 unterbezirk, 'Raub': raub}
73             else:
74                 if raub > max_raub_dict[aktueller_oberbezirk]['Raub']:
75                     max_raub_dict[aktueller_oberbezirk] = {'Unterbezirk':
76 unterbezirk, 'Raub': raub}
77
78 # Ergebnisse ausgeben
79 print("Unterbezirk mit den meisten Raubdelikten pro Oberbezirk:")
80 for oberbezirk in oberbezirke:
81     if oberbezirk in max_raub_dict:
82         unterbezirk = max_raub_dict[oberbezirk]['Unterbezirk']
83         raub = max_raub_dict[oberbezirk]['Raub']
84         print(f"{oberbezirk}: {unterbezirk} ({raub} Raubdelikte)")
85     else:
86         print(f"{oberbezirk}: Keine Unterbezirke gefunden oder keine Daten
87 vorhanden.")
88
89 """
90
91 ### Erklärung des Skripts

```

```

87
88 1. **Import der notwendigen Bibliotheken:**
89   - 'pandas' wird verwendet, um die Excel-Datei zu lesen und die Daten zu
    verarbeiten.
90
91 2. **Festlegen der Excel-Datei und des Sheets:**
92   - Der Pfad zur Excel-Datei und der Name des Sheets werden definiert.
93
94 3. **Definieren der Oberbezirke:**
95   - Eine Liste aller Oberbezirke wird erstellt, um diese während der
    Iteration zu erkennen.
96
97 4. **Daten einlesen und aufbereiten:**
98   - Die Excel-Datei wird eingelesen.
99   - Die letzten zwei Zeilen, die Gesamtwerte enthalten, werden entfernt.
100  - Die Spaltennamen werden bereinigt, um mögliche Leerzeichen zu entfernen.
101
102 5. **Durchgehen der Daten:**
103   - Das Skript iteriert durch jede Zeile des DataFrames.
104   - Wenn ein Oberbezirk erkannt wird, wird dieser als aktueller Oberbezirk
    gespeichert.
105   - Für jede Unterbezirk-Zeile werden die Anzahl der Raubdelikte überprüft
    und das Maximum pro Oberbezirk gespeichert.
106
107 6. **Ausgabe der Ergebnisse:**
108   - Am Ende werden die Unterbezirke mit den meisten Raubdelikten für jeden
    Oberbezirk ausgegeben.
109
110 ### Beispielhafte Ausgabe
111
112 '''
113 Unterbezirk mit den meisten Raubdelikten pro Oberbezirk:
114 Mitte: Alexanderplatz (199 Raubdelikte)
115 Friedrichshain-Kreuzberg: Tiergarten Süd (47 Raubdelikte)
116 Pankow: Regierungsviertel (44 Raubdelikte)
117 Charlottenburg-Wilmersdorf: ... (Raubdelikte)
118 ...
119 '''
120
121 ### Hinweise
122
123 - Stellen Sie sicher, dass die Spaltennamen in Ihrer Excel-Datei genau mit
    den im Skript verwendeten Namen übereinstimmen. Falls Unterschiede
    bestehen (z.B. zusätzliche Leerzeichen), passen Sie die Spaltennamen im
    Skript entsprechend an.
124 - Das Skript ignoriert die letzten zwei Zeilen der Tabelle, wie von Ihnen gew
    ünscht.
125 - Falls bestimmte Oberbezirke keine Unterbezirke haben oder keine Daten
    vorhanden sind, wird dies entsprechend in der Ausgabe vermerkt.
126
127 Ich hoffe, dieses Skript hilft Ihnen weiter! Bei weiteren Fragen stehe ich
    gerne zur Verfügung.

```

Listing 168: API Response Testfall 5 Prompt 2 Ausführung 1

```
1 Hier ist ein Python-Skript, das die gewünschten Anforderungen erfüllt. Das
  Skript verwendet die 'pandas'-Bibliothek, um die Excel-Datei zu lesen und
  die Daten zu verarbeiten. Es identifiziert die Oberbezirke, sammelt die
  zugehörigen Unterbezirke und ermittelt für jeden Oberbezirk den
  Unterbezirk mit den meisten Raubdelikten.
2
3 Stellen Sie sicher, dass Sie die 'pandas'-Bibliothek installiert haben. Falls
  nicht, können Sie sie mit folgendem Befehl 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 # Pfad zur Excel-Datei
15 excel_file = 'Fallzahlen&HZ 2014-2023.xlsx'
16
17 # Name des Sheets
18 sheet_name = 'Fallzahlen_2023'
19
20 # Liste der Oberbezirke
21 oberbezirke = [
22     'Mitte',
23     'Friedrichshain-Kreuzberg',
24     'Pankow',
25     'Charlottenburg-Wilmersdorf',
26     'Spandau',
27     'Steglitz-Zehlendorf',
28     'Tempelhof-Schöneberg',
29     'Neukölln',
30     'Treptow-Köpenick',
31     'Marzahn-Hellersdorf',
32     'Lichtenberg',
33     'Reinickendorf'
34 ]
35
36 # Lesen des Excel-Sheets
37 df = pd.read_excel(excel_file, sheet_name=sheet_name)
38
39 # Entfernen der letzten zwei Zeilen (Gesamtwerte)
40 df = df.iloc[:-2].reset_index(drop=True)
41
42 # Initialisierung
43 current_oberbezirk = None
44 unterbezirke_dict = {ober: [] for ober in oberbezirke}
```

```

45
46 # Iteration über die Zeilen des DataFrames
47 for index, row in df.iterrows():
48     bezirk_name = row['Bezeichnung (Bezirksregion)']
49
50     if bezirk_name in oberbezirke:
51         # Aktueller Oberbezirk festlegen
52         current_oberbezirk = bezirk_name
53     else:
54         if current_oberbezirk:
55             # Unterbezirk zuweisen
56             unterbezirke_dict[current_oberbezirk].append({
57                 'Bezeichnung': bezirk_name,
58                 'Raub': row['Raub']
59             })
60         else:
61             # Falls kein Oberbezirk festgelegt ist, überspringen
62             continue
63
64 # Ermittlung des Unterbezirks mit den meisten Raubdelikten für jeden
65 # Oberbezirk
66 top_unterbezirke = {}
67
68 for oberbezirk, unterbezirke in unterbezirke_dict.items():
69     if unterbezirke:
70         # Finden des Unterbezirks mit dem maximalen 'Raub' Wert
71         top_unterbezirk = max(unterbezirke, key=lambda x: x['Raub'])
72         top_unterbezirke[oberbezirk] = {
73             'Unterbezirk': top_unterbezirk['Bezeichnung'],
74             'Raubdelikte': top_unterbezirk['Raub']
75         }
76     else:
77         top_unterbezirke[oberbezirk] = {
78             'Unterbezirk': None,
79             'Raubdelikte': 0
80         }
81
82 # Ausgabe der Ergebnisse
83 print("Unterbezirk mit den meisten Raubdelikten pro Oberbezirk (2023):\n")
84 for oberbezirk, daten in top_unterbezirke.items():
85     unterbezirk = daten['Unterbezirk']
86     raub = daten['Raubdelikte']
87     if unterbezirk:
88         print(f"{oberbezirk}: {unterbezirk} ({raub} Raubdelikte)")
89     else:
90         print(f"{oberbezirk}: Kein Unterbezirk gefunden.")
91
92 '''
93 ### Erklärung des Skripts
94
95 1. **Importieren der Bibliothek**:
```

```

96     - 'pandas' wird für die Datenverarbeitung verwendet.
97
98 2. **Festlegen der Dateipfade und Namen**:
99     - 'excel_file' definiert den Namen der Excel-Datei.
100    - 'sheet_name' gibt das zu lesende Sheet an.
101
102 3. **Definieren der Oberbezirke**:
103     - Die Liste 'oberbezirke' enthält die Namen der Oberbezirke.
104
105 4. **Lesen und Vorbereiten der Daten**:
106     - Das Excel-Sheet wird gelesen und die letzten zwei Zeilen, die
107       Gesamtwerte enthalten, werden entfernt.
108
109 5. **Zuweisen der Unterbezirke zu den Oberbezirken**:
110     - Das Skript durchläuft jede Zeile und bestimmt, ob die aktuelle Zeile
111       einen Oberbezirk oder einen Unterbezirk darstellt.
112     - Unterbezirke werden dem aktuell festgelegten Oberbezirk zugewiesen.
113
114 6. **Ermitteln des Unterbezirks mit den meisten Raubdelikten**:
115     - Für jeden Oberbezirk wird der Unterbezirk mit dem höchsten Wert in der
116       Spalte 'Raub' identifiziert.
117
118 7. **Ausgabe der Ergebnisse**:
119     - Die Ergebnisse werden in der Konsole ausgegeben, wobei für jeden
120       Oberbezirk der entsprechende Unterbezirk und die Anzahl der Raubdelikte
121       angezeigt werden.
122
123 ### Ausführen des Skripts
124
125 1. Stellen Sie sicher, dass sich die Excel-Datei 'Fallzahlen&HZ 2014-2023.
126    xlsx' im gleichen Verzeichnis wie das Skript befindet. Andernfalls passen
127    Sie den Pfad in der Variable 'excel_file' entsprechend an.
128
129 2. Führen Sie das Skript mit Python aus:
130
131     '''
132     bash
133     python dein_skript_name.py
134     '''
135
136     Ersetzen Sie 'dein_skript_name.py' durch den tatsächlichen Namen Ihres
137     Skripts.
138
139 Nach der Ausführung sollten die Unterbezirke mit den meisten Raubdelikten für
140 jeden Oberbezirk in der Konsole angezeigt werden. Zum Beispiel:
141
142 '''
143 Unterbezirk mit den meisten Raubdelikten pro Oberbezirk (2023):
144
145 Mitte: Regierungsviertel (44 Raubdelikte)
146 Friedrichshain-Kreuzberg: [Unterbezirk] ([Raubzahl] Raubdelikte)
147 ...
148 '''

```

139

140 Passen Sie das Skript bei Bedarf weiter an, um zusätzliche Funktionen oder eine andere Ausgabeform zu integrieren.

Listing 169: API Response Testfall 5 Prompt 2 Ausführung 2

```

1 Hier ist ein Python-Skript, das die gewünschten Anforderungen erfüllt. Das
  Skript verwendet die 'pandas' Bibliothek, um die Excel-Datei zu
  verarbeiten. Es liest das spezifizierte Sheet ein, identifiziert die Ober-
  und Unterbezirke, und ermittelt für jeden Oberbezirk den Unterbezirk mit
  den meisten Raubdelikten.
2
3 **Voraussetzungen:**
4
5 - **Python** sollte installiert sein (Version 3.6 oder höher empfohlen).
6 - Die Bibliothek **pandas** muss installiert sein. Falls nicht, kannst du sie
  mit folgendem Befehl installieren:
7
8   '''bash
9   pip install pandas
10  '''
11
12 - Die Bibliothek **openpyxl** wird benötigt, um Excel-Dateien zu lesen.
  Installiere sie ebenfalls mit:
13
14   '''bash
15   pip install openpyxl
16  '''
17
18 **Python-Skript:**
19
20   '''python
21   import pandas as pd
22
23   # Pfad zur Excel-Datei
24   excel_datei = 'Fallzahlen&HZ 2014-2023.xlsx'
25
26   # Name des Sheets
27   sheet_name = 'Fallzahlen_2023'
28
29   # Liste der Oberbezirke
30   oberbezirke = [
31       'Mitte',
32       'Friedrichshain-Kreuzberg',
33       'Pankow',
34       'Charlottenburg-Wilmersdorf',
35       'Spandau',
36       'Steglitz-Zehlendorf',
37       'Tempelhof-Schöneberg',
38       'Neukölln',
39       'Treptow-Köpenick',
40       'Marzahn-Hellersdorf',
41       'Lichtenberg',

```

```

42     'Reinickendorf'
43 ]
44
45 try:
46     # Excel-Datei einlesen
47     df = pd.read_excel(excel_datei, sheet_name=sheet_name)
48
49     # Entfernen der letzten zwei Zeilen (Gesamtwerte)
50     df = df.iloc[:-2]
51
52     # Sicherstellen, dass die relevanten Spalten vorhanden sind
53     erforderliche_spalten = ['Bezeichnung (Bezirksregion)', 'Raub']
54     for spalte in erforderliche_spalten:
55         if spalte not in df.columns:
56             raise ValueError(f"Die erforderliche Spalte '{spalte}' wurde
nicht gefunden.")
57
58     # Erstellen einer neuen Spalte 'Oberbezirk', die den aktuellen Oberbezirk
enthält
59     df['Oberbezirk'] = df['Bezeichnung (Bezirksregion)'].where(df['
Bezeichnung (Bezirksregion)'].isin(oberbezirke))
60     df['Oberbezirk'] = df['Oberbezirk'].ffill()
61
62     # Filtern der Unterbezirke (Ausschluss der Oberbezirke)
63     unterbezirke = df[~df['Bezeichnung (Bezirksregion)'].isin(oberbezirke)].
copy()
64
65     # Um sicherzustellen, dass die 'Raub' Spalte numerisch ist
66     unterbezirke['Raub'] = pd.to_numeric(unterbezirke['Raub'], errors='coerce
').fillna(0).astype(int)
67
68     # Gruppieren nach 'Oberbezirk' und Finden des Unterbezirks mit den
meisten Raubdelikten
69     idx_max_raub = unterbezirke.groupby('Oberbezirk')['Raub'].idxmax()
70     max_raub_unterbezirke = unterbezirke.loc[idx_max_raub]
71
72     # Ausgabe der Ergebnisse
73     print("Unterbezirk mit den meisten Raubdelikten pro Oberbezirk (2023):\n"
)
74     for _, row in max_raub_unterbezirke.iterrows():
75         print(f"Oberbezirk: {row['Oberbezirk']}")
76         print(f"    Unterbezirk: {row['Bezeichnung (Bezirksregion)']}")
77         print(f"    Anzahl Raubdelikte: {row['Raub']}\n")
78
79 except FileNotFoundError:
80     print(f"Die Datei '{excel_datei}' wurde nicht gefunden. Bitte überprüfe
den Pfad.")
81 except ValueError as ve:
82     print(f"Fehler: {ve}")
83 except Exception as e:
84     print(f"Ein unerwarteter Fehler ist aufgetreten: {e}")
85 '''

```

```

86
87 **Erläuterungen zum Skript:**
88
89 1. **Importieren der Bibliothek:**
90     - 'pandas' wird für die Datenverarbeitung verwendet.
91
92 2. **Einlesen der Excel-Datei:**
93     - Das Skript liest das angegebene Sheet 'Fallzahlen_2023' aus der Excel-
94       Datei 'Fallzahlen&HZ 2014-2023.xlsx'.
95
96 3. **Datenvorbereitung:**
97     - Die letzten zwei Zeilen der Daten, die Gesamtwerte darstellen, werden
98       entfernt.
99     - Es wird überprüft, ob die erforderlichen Spalten ('Bezeichnung (
100       Bezirksregion)' und 'Raub') vorhanden sind.
101
102 4. **Identifikation der Ober- und Unterbezirke:**
103     - Eine neue Spalte 'Oberbezirk' wird erstellt, die den aktuellen
104       Oberbezirk für jede Zeile enthält. Dies wird erreicht, indem die
105       Oberbezirke in der Liste 'oberbezirke' gesucht und dann vorwärts aufgefü-
106       llt werden ('ffill()'), um jedem Unterbezirk seinen zugehörigen Oberbezirk
107       zuzuweisen.
108     - Anschließend werden die Oberbezirke aus dem DataFrame gefiltert, sodass
109       nur die Unterbezirke übrig bleiben.
110
111 5. **Datenbereinigung:**
112     - Die Spalte 'Raub' wird in einen numerischen Datentyp konvertiert, um
113       sicherzustellen, dass die Werte korrekt verarbeitet werden können. Nicht-
114       numerische Werte werden als 0 behandelt.
115
116 6. **Berechnung der Unterbezirke mit den meisten Raubdelikten:**
117     - Das Skript gruppiert die Daten nach 'Oberbezirk' und findet innerhalb
118       jeder Gruppe den Unterbezirk mit der höchsten Anzahl an Raubdelikten.
119
120 7. **Ausgabe der Ergebnisse:**
121     - Für jeden Oberbezirk wird der entsprechende Unterbezirk mit den meisten
122       Raubdelikten sowie die Anzahl der Raubdelikte ausgegeben.
123
124 8. **Fehlerbehandlung:**
125     - Das Skript enthält grundlegende Fehlerbehandlungen, um fehlende Dateien
126       oder Spalten zu erkennen und unerwartete Fehler abzufangen.
127
128 **Anwendung:**
129
130 1. Stelle sicher, dass sich die Excel-Datei 'Fallzahlen&HZ 2014-2023.xlsx'
131    im gleichen Verzeichnis wie das Skript befindet oder passe den Pfad
132    entsprechend an.
133
134 2. Führe das Skript mit Python aus:
135
136     'bash
137     python dein_skript_name.py

```

```

123     '''
124
125 3. Das Skript gibt die Ergebnisse in der Konsole aus, zum Beispiel:
126
127     '''
128     Unterbezirk mit den meisten Raubdelikten pro Oberbezirk (2023):
129
130     Oberbezirk: Mitte
131         Unterbezirk: Alexanderplatz
132         Anzahl Raubdelikte: 199
133
134     Oberbezirk: Friedrichshain-Kreuzberg
135         Unterbezirk: ...
136         Anzahl Raubdelikte: ...
137
138     ...
139     '''
140
141 **Hinweis:**
142
143 - Stelle sicher, dass die Spaltennamen in der Excel-Datei genau mit denen im
    Skript übereinstimmen. Falls es Abweichungen gibt, passe die Spaltennamen
    im Skript entsprechend an.
144 - Bei größeren Datenmengen oder speziellen Anforderungen kann das Skript
    weiter angepasst werden.

```

Listing 170: API Response Testfall 5 Prompt 2 Ausführung 3

```

1 Natürlich! Hier ist ein Python-Skript, das die gewünschten Anforderungen erfü
  llt. Dieses Skript verwendet das 'pandas'-Paket, um die Excel-Datei zu
  verarbeiten. Stelle sicher, dass du 'pandas' und 'openpyxl' installiert
  hast, bevor du das Skript ausführst. Du kannst sie mit 'pip' installieren,
  falls sie noch nicht installiert sind:
2
3 '''bash
4 pip install pandas openpyxl
5 '''
6
7 ### Python-Skript
8
9 '''python
10 import pandas as pd
11
12 def find_top_raub_unterbezirke(excel_file, sheet_name):
13     """
14     Liest die Daten aus dem angegebenen Excel-Sheet und ermittelt für jeden
15     Oberbezirk
16     den Unterbezirk mit den meisten Raubdelikten.
17
18     :param excel_file: Pfad zur Excel-Datei
19     :param sheet_name: Name des Sheets, das ausgelesen werden soll
20     :return: Dictionary mit Oberbezirken als Schlüsseln und den
21             entsprechenden

```

```

20         Unterbezirken mit den meisten Raubdelikten als Werten
21     """
22     # Liste der Oberbezirke
23     oberbezirke = [
24         'Mitte',
25         'Friedrichshain-Kreuzberg',
26         'Pankow',
27         'Charlottenburg-Wilmersdorf',
28         'Spandau',
29         'Steglitz-Zehlendorf',
30         'Tempelhof-Schöneberg',
31         'Neukölln',
32         'Treptow-Köpenick',
33         'Marzahn-Hellersdorf',
34         'Lichtenberg',
35         'Reinickendorf'
36     ]
37
38     try:
39         # Excel-Datei einlesen
40         df = pd.read_excel(excel_file, sheet_name=sheet_name, engine='
openpyxl')
41     except FileNotFoundError:
42         print(f"Fehler: Die Datei '{excel_file}' wurde nicht gefunden.")
43         return
44     except Exception as e:
45         print(f"Ein Fehler ist aufgetreten: {e}")
46         return
47
48     # Die letzten zwei Zeilen entfernen (Gesamtwerte)
49     df = df.iloc[:-2].reset_index(drop=True)
50
51     # Initialisiere ein Dictionary, um die Unterbezirke für jeden Oberbezirk
zu speichern
52     oberbezirk_dict = {ober: [] for ober in oberbezirke}
53
54     current_oberbezirk = None
55
56     # Durchlaufe jede Zeile des DataFrames
57     for idx, row in df.iterrows():
58         bezirk_name = row['Bezeichnung (Bezirksregion)']
59
60         if bezirk_name in oberbezirke:
61             # Aktuellen Oberbezirk setzen
62             current_oberbezirk = bezirk_name
63             continue # Weiter zur nächsten Zeile
64
65         # Wenn kein aktueller Oberbezirk gesetzt ist, überspringe die Zeile
66         if current_oberbezirk is None:
67             continue
68
69         # Extrahiere den Raub-Wert

```



```

70     raub_count = row.get('Raub')
71
72     # Überprüfen, ob der Raub-Wert gültig ist
73     if pd.isna(raub_count):
74         raub_count = 0 # Oder eine andere Standardbehandlung
75
76     # Füge den Unterbezirk und den Raub-Wert dem aktuellen Oberbezirk
77     hinzu
78     unterbezirk = {
79         'Name': bezirk_name,
80         'Raub': raub_count
81     }
82     oberbezirk_dict[current_oberbezirk].append(unterbezirk)
83
84     # Ergebnis-Dictionary, das die Top-Unterbezirke speichert
85     top_unterbezirke = {}
86
87     for ober, unterbezirke in oberbezirk_dict.items():
88         if not unterbezirke:
89             top_unterbezirke[ober] = None
90             continue
91
92         # Finde den Unterbezirk mit den meisten Raubdelikten
93         top_unter = max(unterbezirke, key=lambda x: x['Raub'])
94         top_unterbezirke[ober] = top_unter
95
96     return top_unterbezirke
97
98 def main():
99     # Pfad zur Excel-Datei
100     excel_file = 'Fallzahlen&HZ 2014-2023.xlsx'
101
102     # Name des Sheets
103     sheet_name = 'Fallzahlen_2023'
104
105     # Finde die Top-Unterbezirke
106     top_unterbezirke = find_top_raub_unterbezirke(excel_file, sheet_name)
107
108     if not top_unterbezirke:
109         print("Keine Daten gefunden oder ein Fehler ist aufgetreten.")
110         return
111
112     # Ausgabe der Ergebnisse
113     print("Unterbezirke mit den meisten Raubdelikten pro Oberbezirk:\n")
114     for ober, unter in top_unterbezirke.items():
115         if unter:
116             print(f"Oberbezirk: {ober}")
117             print(f"    Unterbezirk: {unter['Name']}")
118             print(f"    Raubdelikte: {unter['Raub']}\n")
119         else:
120             print(f"Oberbezirk: {ober} hat keine Unterbezirke oder keine
121 Raubdelikte.\n")

```

```

120
121 if __name__ == "__main__":
122     main()
123 '''
124
125 ### Erklärung des Skripts
126
127 1. **Importieren der notwendigen Bibliotheken:**
128     - 'pandas' wird verwendet, um die Excel-Datei zu lesen und zu verarbeiten.
129
130 2. **Definieren der Funktion 'find_top_raub_unterbezirke':**
131     - Diese Funktion liest die Excel-Datei und das angegebene Sheet.
132     - Sie entfernt die letzten zwei Zeilen, die Gesamtwerte enthalten.
133     - Sie iteriert durch jede Zeile und identifiziert Ober- und Unterbezirke.
134     - Für jeden Oberbezirk werden die Unterbezirke und deren Raubdelikte
135       gesammelt.
136     - Am Ende wird für jeden Oberbezirk der Unterbezirk mit den meisten
137       Raubdelikten ermittelt.
138
139 3. **Definieren der 'main'-Funktion:**
140     - Setzt den Pfad zur Excel-Datei und den Namen des Sheets.
141     - Ruft die Funktion auf, um die Top-Unterbezirke zu finden.
142     - Gibt die Ergebnisse übersichtlich aus.
143
144 4. **Ausführen des Skripts:**
145     - Das Skript startet mit der 'main'-Funktion, wenn es direkt ausgeführt
146       wird.
147
148 ### Hinweise
149
150 - **Spaltennamen:** Stelle sicher, dass die Spaltennamen in der Excel-Datei
151   genau mit den im Skript verwendeten Namen übereinstimmen, insbesondere '
152   Bezeichnung (Bezirksregion)' und 'Raub'. Wenn die Spaltennamen
153   Abweichungen aufweisen (z.B. zusätzliche Leerzeichen oder Zeilenumbrüche),
154   musst du diese entsprechend anpassen.
155
156 - **Fehlerbehandlung:** Das Skript enthält grundlegende Fehlerbehandlungen, z
157   .B. wenn die Datei nicht gefunden wird oder andere Lese Probleme auftreten.
158
159 - **Leistungsoptimierung:** Für sehr große Dateien kann es sinnvoll sein,
160   Optimierungen vorzunehmen, aber für die gegebene Datenmenge (169 Zeilen)
161   sollte das Skript schnell genug sein.
162
163 ### Beispielausgabe
164
165 Nach dem Ausführen des Skripts könnte die Ausgabe wie folgt aussehen:
166
167 '''
168 Unterbezirke mit den meisten Raubdelikten pro Oberbezirk:
169
170 Oberbezirk: Mitte
171 Unterbezirk: Alexanderplatz

```

```

162 Raubdelikte: 199
163
164 Oberbezirk: Friedrichshain-Kreuzberg
165 Unterbezirk: Tiergarten Süd
166 Raubdelikte: 47
167
168 Oberbezirk: Pankow
169 Unterbezirk: Regierungsviertel
170 Raubdelikte: 44
171
172 ... (weitere Oberbezirke)
173
174 '''
175
176 Diese Ausgabe zeigt für jeden Oberbezirk den Unterbezirk mit den meisten
177 Raubdelikten sowie die Anzahl der Raubdelikte in diesem Unterbezirk.
178
179 ### Anpassungen
180
181 Falls du die Ergebnisse in eine Datei schreiben möchtest (z.B. CSV), kannst
182 du das Skript entsprechend erweitern. Hier ein kurzes Beispiel, wie du die
183 Ergebnisse in eine CSV-Datei speichern kannst:
184
185 '''python
186 import csv
187
188 def save_results_to_csv(top_unterbezirke, output_file):
189     """
190     Speichert die Ergebnisse in eine CSV-Datei.
191
192     :param top_unterbezirke: Dictionary mit den Ergebnissen
193     :param output_file: Pfad zur Ausgabedatei
194     """
195     with open(output_file, mode='w', newline='', encoding='utf-8') as csvfile:
196         fieldnames = ['Oberbezirk', 'Unterbezirk', 'Raubdelikte']
197         writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
198
199         writer.writeheader()
200         for ober, unter in top_unterbezirke.items():
201             if unter:
202                 writer.writerow({
203                     'Oberbezirk': ober,
204                     'Unterbezirk': unter['Name'],
205                     'Raubdelikte': unter['Raub']
206                 })
207             else:
208                 writer.writerow({
209                     'Oberbezirk': ober,
210                     'Unterbezirk': 'Keine Daten',
211                     'Raubdelikte': 0
212                 })

```

```

210
211 # In der main-Funktion nach dem Ausgeben der Ergebnisse hinzufügen:
212 # Speichern in CSV
213 output_csv = 'Top_Unterbezirke_Raubdelikte.csv'
214 save_results_to_csv(top_unterbezirke, output_csv)
215 print(f"Die Ergebnisse wurden in '{output_csv}' gespeichert.")
216 '''
217
218 Füge diese Funktion hinzu und rufe sie in der 'main'-Funktion nach der
    Ausgabe der Ergebnisse auf, um die Resultate auch in eine CSV-Datei zu
    speichern.
219
220 Ich hoffe, dieses Skript hilft dir weiter! Wenn du weitere Anpassungen oder
    Erklärungen benötigst, lass es mich wissen.

```

Listing 171: API Response Testfall 5 Prompt 2 Ausführung 4

```

1 Hier ist ein Python-Skript, das die gewünschten Anforderungen erfüllt. Es
    verwendet die Bibliothek 'pandas', um die Excel-Datei zu lesen und die
    Daten zu verarbeiten. Stellen Sie sicher, dass Sie 'pandas' installiert
    haben ('pip install pandas') und dass die Bibliothek 'openpyxl' für das
    Lesen von '.xlsx'-Dateien installiert ist ('pip install openpyxl').
2
3 '''python
4 import pandas as pd
5
6 # Definiere den Dateipfad und den Sheet-Namen
7 excel_datei = 'Fallzahlen&HZ 2014-2023.xlsx'
8 sheet_name = 'Fallzahlen_2023'
9
10 # Liste der Oberbezirke
11 oberbezirke = [
12     'Mitte',
13     'Friedrichshain-Kreuzberg',
14     'Pankow',
15     'Charlottenburg-Wilmersdorf',
16     'Spandau',
17     'Steglitz-Zehlendorf',
18     'Tempelhof-Schöneberg',
19     'Neukölln',
20     'Treptow-Köpenick',
21     'Marzahn-Hellersdorf',
22     'Lichtenberg',
23     'Reinickendorf'
24 ]
25
26 # Lese das Excel-Sheet ein
27 df = pd.read_excel(excel_datei, sheet_name=sheet_name)
28
29 # Entferne die letzten zwei Zeilen (Gesamtwerte)
30 df = df.iloc[:-2].reset_index(drop=True)
31
32 # Bereinige die Spaltennamen (entferne Zeilenumbrüche und Leerzeichen)

```

```

33 df.columns = df.columns.str.replace('\n', ' ').str.strip()
34
35 # Stelle sicher, dass die 'Raub' Spalte numerisch ist
36 df['Raub'] = pd.to_numeric(df['Raub'], errors='coerce')
37
38 # Initialisiere ein Dictionary, um die Ergebnisse zu speichern
39 ergebnisse = {}
40
41 # Variable, um den aktuellen Oberbezirk zu verfolgen
42 aktueller_oberbezirk = None
43
44 # Iteriere über die DataFrame-Zeilen
45 for index, row in df.iterrows():
46     bezirk_bezeichnung = row['Bezeichnung (Bezirksregion)']
47
48     if bezirk_bezeichnung in oberbezirke:
49         # Aktuellen Oberbezirk setzen
50         aktueller_oberbezirk = bezirk_bezeichnung
51         # Initialisiere eine Liste für die Unterbezirke des aktuellen
Oberbezirks
52         ergebnisse[aktueller_oberbezirk] = []
53     else:
54         if aktueller_oberbezirk:
55             # Füge den Unterbezirk zu der Liste des aktuellen Oberbezirks
hinzu
56             ergebnisse[aktueller_oberbezirk].append({
57                 'Unterbezirk': bezirk_bezeichnung,
58                 'Raub': row['Raub']
59             })
60
61 # Jetzt finde für jeden Oberbezirk den Unterbezirk mit den meisten
Raubdelikten
62 for oberbezirk, unterbezirk_list in ergebnisse.items():
63     if not unterbezirk_list:
64         print(f"Oberbezirk '{oberbezirk}' hat keine Unterbezirke.")
65         continue
66
67     # Finde den Unterbezirk mit dem maximalen Raub-Wert
68     max_raub_unterbezirk = max(unterbezirk_list, key=lambda x: x['Raub'])
69
70     print(f"Oberbezirk: {oberbezirk}")
71     print(f"  Unterbezirk mit den meisten Raubdelikten: {max_raub_unterbezirk
['Unterbezirk']} ({max_raub_unterbezirk['Raub']} Raubdelikte)")
72     print()
73
74 '''
75
76 ### Erklärung des Skripts
77
78 1. **Bibliotheken importieren:**
79     Das Skript verwendet 'pandas' zur Datenverarbeitung.
80

```

```

81 2. **Dateipfad und Sheet-Name definieren:**
82     Stellen Sie sicher, dass die Excel-Datei im selben Verzeichnis wie das
      Skript liegt oder geben Sie den vollständigen Pfad an.
83
84 3. **Oberbezirke definieren:**
85     Eine Liste der Oberbezirke wird erstellt, um sie später im DataFrame zu
      identifizieren.
86
87 4. **Excel-Datei einlesen:**
88     Das angegebene Sheet wird in einen DataFrame geladen.
89
90 5. **Letzte zwei Zeilen entfernen:**
91     Diese enthalten die Gesamtwerte und sollen ignoriert werden.
92
93 6. **Spaltennamen bereinigen:**
94     Entfernt Zeilenumbrüche und überflüssige Leerzeichen aus den Spaltennamen,
      um sicherzustellen, dass sie korrekt referenziert werden können.
95
96 7. **'Raub' Spalte in numerisch umwandeln:**
97     Dies stellt sicher, dass die Werte für die Berechnung als Zahlen behandelt
      werden.
98
99 8. **Daten strukturieren:**
100    Das Skript durchläuft jede Zeile, identifiziert Oberbezirke und sammelt
      die entsprechenden Unterbezirke mit ihren 'Raub'-Werten.
101
102 9. **Unterbezirk mit den meisten Raubdelikten finden:**
103    Für jeden Oberbezirk wird der Unterbezirk mit dem höchsten 'Raub'-Wert
      ermittelt und ausgegeben.
104
105 ### Beispielausgabe
106
107     '''
108     Oberbezirk: Mitte
109         Unterbezirk mit den meisten Raubdelikten: Regierungsviertel (44 Raubdelikte
110         )
111
112     Oberbezirk: Friedrichshain-Kreuzberg
113         Unterbezirk mit den meisten Raubdelikten: Alexanderplatz (199 Raubdelikte)
114     ...
115     '''
116
117 ### Hinweise
118
119 - **Fehlerbehandlung:** Das Skript geht davon aus, dass die Daten konsistent
      sind. Für eine robustere Lösung könnten zusätzliche Überprüfungen und
      Fehlerbehandlungen hinzugefügt werden.
120 - **Leistungsoptimierung:** Für sehr große Datensätze könnte die Leistung
      optimiert werden, aber für die gegebene Anzahl von Zeilen ist dies nicht
      erforderlich.
121 - **Weitere Anpassungen:** Wenn zusätzliche Anforderungen bestehen (z.B.

```

```

Export der Ergebnisse in eine Datei), kann das Skript entsprechend
erweitert werden.
122
123 ### Ausführung des Skripts
124
125 1. Stellen Sie sicher, dass Python installiert ist.
126
127 2. Installieren Sie die erforderlichen Bibliotheken, falls noch nicht
    geschehen:
128
129     '''bash
130     pip install pandas openpyxl
131     '''
132
133 3. Speichern Sie das Skript in einer '.py'-Datei, z.B. 'raubdelikte.py'.
134
135 4. Führen Sie das Skript in der Kommandozeile oder einer Python-Umgebung aus:
136
137     '''bash
138     python raubdelikte.py
139     '''
140
141 Das Skript gibt dann für jeden Oberbezirk den Unterbezirk mit den meisten
    Raubdelikten aus.

```

Listing 172: API Response Testfall 5 Prompt 2 Ausführung 5

```

1 Gerne erstelle ich Ihnen ein Python-Skript, das die beschriebenen Schritte
    ausführt. Wir werden die Bibliothek 'pandas' verwenden, um die Excel-Datei
    zu verarbeiten. Stellen Sie sicher, dass Sie 'pandas' und 'openpyxl'
    installiert haben. Sie können dies mit den folgenden Befehlen tun:
2
3 '''bash
4 pip install pandas openpyxl
5 '''
6
7 Hier ist das vollständige Skript mit ausführlichen Kommentaren:
8
9 '''python
10 import pandas as pd
11
12 # Schritt 1: Excel-Datei und Sheet definieren
13 file_path = 'Fallzahlen&HZ 2014-2023.xlsx'
14 sheet_name = 'Fallzahlen_2023'
15
16 # Schritt 2: Daten aus dem Excel-Sheet in ein Pandas DataFrame laden
17 try:
18     df = pd.read_excel(file_path, sheet_name=sheet_name)
19 except FileNotFoundError:
20     print(f"Die Datei {file_path} wurde nicht gefunden.")
21     exit(1)
22 except ValueError:
23     print(f"Das Sheet '{sheet_name}' existiert nicht in der Datei {file_path}

```

```

    }.)
24     exit(1)
25
26 # Schritt 3: Notwendige Spalten auswählen
27 try:
28     df = df[['Bezeichnung (Bezirksregion)', 'Raub']]
29 except KeyError as e:
30     print(f"Die erforderliche Spalte wurde nicht gefunden: {e}")
31     exit(1)
32
33 # Schritt 4: Die letzten zwei Zeilen entfernen (Gesamtwerte)
34 df = df.iloc[:-2]
35
36 # Schritt 5: Liste der Oberbezirke definieren
37 oberbezirke = [
38     'Mitte',
39     'Friedrichshain-Kreuzberg',
40     'Pankow',
41     'Charlottenburg-Wilmersdorf',
42     'Spandau',
43     'Steglitz-Zehlendorf',
44     'Tempelhof-Schöneberg',
45     'Neukölln',
46     'Treptow-Köpenick',
47     'Marzahn-Hellersdorf',
48     'Lichtenberg',
49     'Reinickendorf'
50 ]
51
52 # Schritt 6: Oberbezirke identifizieren und Unterbezirke zuweisen
53 # Eine neue Spalte 'Oberbezirk' erstellen, die nur dann gesetzt ist, wenn die
54 # Zeile ein Oberbezirk ist
55 df['Oberbezirk'] = df['Bezeichnung (Bezirksregion)'].where(df['Bezeichnung (
56 # Die 'Oberbezirk'-Spalte nach unten füllen, sodass jeder Unterbezirk seinen
57 # zugehörigen Oberbezirk hat
58 df['Oberbezirk'] = df['Oberbezirk'].ffill()
59
60 # Schritt 7: Unterbezirke aus den Daten filtern (Zeilen, die keine
61 # Oberbezirke sind)
62 df_unterbezirke = df[~df['Bezeichnung (Bezirksregion)'].isin(oberbezirke)].
63     copy()
64
65 # Schritt 8: Für jeden Oberbezirk den Unterbezirk mit dem höchsten 'Raub'
66 # ermitteln
67 # Gruppieren nach 'Oberbezirk' und Finden des Index mit maximalem 'Raub' in
68 # jeder Gruppe
69 idx = df_unterbezirke.groupby('Oberbezirk')['Raub'].idxmax()
70
71 # Das finale DataFrame erstellen mit den gewünschten Spalten
72 result_df = df_unterbezirke.loc[idx, ['Oberbezirk', 'Bezeichnung (

```



```

        Bezirksregion)', 'Raub']] .copy()
68
69 # Spalten umbenennen für bessere Lesbarkeit
70 result_df.rename(columns={'Bezeichnung (Bezirksregion)': 'Unterbezirk'},
71                    inplace=True)
72
73 # Schritt 9: Das finale DataFrame ausgeben
74 print(result_df.reset_index(drop=True))
75
76 ### Erklärung der Schritte:
77
78 1. **Importieren der Bibliotheken:**
79     - 'pandas' wird für die Datenverarbeitung genutzt.
80
81 2. **Laden der Excel-Datei:**
82     - Das Skript versucht, die angegebene Excel-Datei und das Sheet zu laden.
83       Fehler beim Laden der Datei oder des Sheets werden abgefangen und
84       angezeigt.
85
86 3. **Auswahl der relevanten Spalten:**
87     - Es werden nur die Spalten 'Bezeichnung (Bezirksregion)' und 'Raub'
88       ausgewählt. Wird eine dieser Spalten nicht gefunden, gibt das Skript eine
89       Fehlermeldung aus und beendet sich.
90
91 4. **Entfernen der Gesamtwerte:**
92     - Die letzten zwei Zeilen, die Gesamtwerte enthalten, werden entfernt.
93
94 5. **Definieren der Oberbezirke:**
95     - Eine Liste mit den Namen der Oberbezirke wird erstellt.
96
97 6. **Zuweisen der Oberbezirke zu Unterbezirken:**
98     - Eine neue Spalte 'Oberbezirk' wird erstellt, die nur dann den Namen
99       des Oberbezirks enthält, wenn die Zeile selbst ein Oberbezirk ist.
100     - Mit 'ffill()' werden die Oberbezirksnamen nach unten gefüllt, sodass
101       jeder Unterbezirk seinen zugehörigen Oberbezirk hat.
102
103 7. **Filtern der Unterbezirke:**
104     - Es werden nur die Zeilen berücksichtigt, die keine Oberbezirke sind.
105
106 8. **Ermitteln des Unterbezirks mit dem höchsten 'Raub' pro Oberbezirk:**
107     - Für jeden Oberbezirk wird der Unterbezirk mit dem höchsten Wert in der
108       Spalte 'Raub' identifiziert.
109     - Die Ergebnisse werden in einem neuen DataFrame 'result_df' gespeichert
110       und die Spalten entsprechend umbenannt.
111
112 9. **Ausgabe des finalen DataFrames:**
113     - Das finale DataFrame wird ausgegeben, wobei der Index zurückgesetzt wird
114       für eine saubere Darstellung.
115
116 ### Beispielhafte Ausgabe

```

```

109 Die Ausgabe wird ein DataFrame sein, das so aussieht:
110
111 '''
112
113         Oberbezirk          Unterbezirk    Raub
114 0             Mitte          Unterbezirk1    500
115 1  Friedrichshain-Kreuzberg          Unterbezirk2    450
116 2             Pankow          Unterbezirk3    300
117 ...
118 '''
119 Jede Zeile zeigt den Oberbezirk, den entsprechenden Unterbezirk mit dem hö
    chsten 'Raub'-Wert und den Wert selbst.
120
121 ### Hinweise
122
123 - **Dateipfad anpassen:** Stellen Sie sicher, dass sich die Excel-Datei '
    Fallzahlen&HZ 2014-2023.xlsx' im gleichen Verzeichnis wie das Skript
    befindet oder passen Sie den 'file_path' entsprechend an.
124
125 - **Excel-Sheet prüfen:** Stellen Sie sicher, dass das Sheet '
    Fallzahlen_2023' in Ihrer Excel-Datei existiert und die erwarteten
    Spalten enthält.
126
127 - **Groß-/Kleinschreibung:** Achten Sie darauf, dass die Namen der
    Oberbezirke genau mit denen in der Spalte 'Bezeichnung (Bezirksregion)'
    übereinstimmen. Eventuell müssen Sie Leerzeichen oder Sonderzeichen
    anpassen.
128
129 Dieses Skript sollte die gewünschten Daten korrekt verarbeiten und das finale
    DataFrame mit den Oberbezirken, den entsprechenden Unterbezirken mit den
    höchsten 'Raub'-Werten und diesen Werten selbst anzeigen.

```

Listing 173: API Response Testfall 5 Prompt 3 Ausführung 1

```

1 Hier ist ein Python-Skript, das die von Ihnen beschriebenen Schritte durchfü
    hrt. Das Skript nutzt die 'pandas' Bibliothek, um die Excel-Datei zu
    verarbeiten. Stellen Sie sicher, dass Sie die benötigten Bibliotheken
    installiert haben. Falls nicht, können Sie diese mit 'pip' installieren:
2
3 '''bash
4 pip install pandas openpyxl
5 '''
6
7 ### Python-Skript
8
9 '''python
10 import pandas as pd
11
12 # 1. Die Daten des Sheets 'Fallzahlen_2023' auslesen
13 file_path = 'Fallzahlen&HZ 2014-2023.xlsx'
14 sheet_name = 'Fallzahlen_2023'
15
16 # 2. Die Daten in einem Pandas Dataframe speichern

```

```

17 df = pd.read_excel(file_path, sheet_name=sheet_name)
18
19 # 3. Das Dataframe auf die notwendigen Spalten reduzieren: 'Bezeichnung (
    Bezirksregion)' und 'Raub'
20 df = df[['Bezeichnung (Bezirksregion)', 'Raub']]
21
22 # 4. Unnötige Zeilen entfernen: Die letzten zwei Zeilen der Tabelle sind
    Gesamtwerte und sollen entfernt werden.
23 df = df.iloc[:-2]
24
25 # 5. Die Oberbezirke in der Spalte 'Bezeichnung (Bezirksregion)'
    identifizieren
26 oberbezirke = [
27     'Mitte',
28     'Friedrichshain-Kreuzberg',
29     'Pankow',
30     'Charlottenburg-Wilmersdorf',
31     'Spandau',
32     'Steglitz-Zehlendorf',
33     'Tempelhof-Schöneberg',
34     'Neukölln',
35     'Treptow-Köpenick',
36     'Marzahn-Hellersdorf',
37     'Lichtenberg',
38     'Reinickendorf'
39 ]
40
41 # Erstellen einer neuen Spalte 'Oberbezirk', die den aktuellen Oberbezirk
    enthält
42 df['Oberbezirk'] = df['Bezeichnung (Bezirksregion)'].where(df['Bezeichnung (
    Bezirksregion)'].isin(oberbezirke))
43 df['Oberbezirk'] = df['Oberbezirk'].ffill()
44
45 # 6. Die Unterbezirke identifizieren, indem Oberbezirke selbst ausgeschlossen
    werden
46 df_unter = df[~df['Bezeichnung (Bezirksregion)'].isin(oberbezirke)].copy()
47
48 # Sicherstellen, dass die 'Raub'-Spalte numerisch ist
49 df_unter['Raub'] = pd.to_numeric(df_unter['Raub'], errors='coerce')
50
51 # Entfernen von Zeilen mit fehlenden 'Raub'-Werten
52 df_unter = df_unter.dropna(subset=['Raub'])
53
54 # Für jeden Oberbezirk den Unterbezirk mit dem höchsten Wert in der Spalte '
    Raub' identifizieren
55 idx_max_raub = df_unter.groupby('Oberbezirk')['Raub'].idxmax()
56 df_max_raub = df_unter.loc[idx_max_raub]
57
58 # 7. Die Ergebnisse in einem neuen Dataframe speichern mit den Spalten: '
    Oberbezirk', 'Unterbezirk', 'Raub'
59 final_df = df_max_raub[['Oberbezirk', 'Bezeichnung (Bezirksregion)', 'Raub'
    ]].rename(columns={

```

```

60     'Bezeichnung (Bezirksregion)': 'Unterbezirk'
61 }).reset_index(drop=True)
62
63 # 8. Das finale Dataframe ausgeben
64 print(final_df)
65 '''
66
67 ### Erklärung der Schritte
68
69 1. **Excel-Datei einlesen:**
70     Das Skript liest das Sheet 'Fallzahlen_2023' aus der Excel-Datei '
71     Fallzahlen&HZ 2014-2023.xlsx' ein und speichert die Daten in einem Pandas
72     DataFrame 'df'.
73
74 2. **Relevante Spalten auswählen:**
75     Es werden nur die Spalten 'Bezeichnung (Bezirksregion)' und 'Raub'
76     behalten, da diese für die weitere Verarbeitung benötigt werden.
77
78 3. **Letzte zwei Zeilen entfernen:**
79     Die letzten zwei Zeilen des DataFrames, die Gesamtwerte enthalten, werden
80     entfernt.
81
82 4. **Oberbezirke identifizieren:**
83     Eine Liste 'oberbezirke' enthält alle Oberbezirke. Eine neue Spalte '
84     Oberbezirk' wird erstellt, die den Namen des Oberbezirks enthält. Mit '
85     ffill()' (forward fill) wird der aktuelle Oberbezirk auf die nachfolgenden
86     Unterbezirke angewendet, bis ein neuer Oberbezirk gefunden wird.
87
88 5. **Unterbezirke filtern:**
89     Zeilen, die selbst Oberbezirke sind, werden ausgeschlossen, sodass nur die
90     Unterbezirke verbleiben.
91
92 6. **Daten bereinigen:**
93     - Die Spalte 'Raub' wird in numerische Werte umgewandelt. Nicht-
94     konvertierbare Werte werden als 'NaN' gesetzt.
95     - Zeilen mit fehlenden 'Raub'-Werten werden entfernt.
96
97 7. **Unterbezirk mit höchstem Raub je Oberbezirk:**
98     Für jeden Oberbezirk wird der Unterbezirk mit dem höchsten Wert in der
99     Spalte 'Raub' identifiziert. Das Ergebnis wird in 'final_df' gespeichert
100    , das die Spalten 'Oberbezirk', 'Unterbezirk' und 'Raub' enthält.
101
102 8. **Ergebnis anzeigen:**
103     Das finale DataFrame wird ausgegeben.
104
105 ### Beispielausgabe
106
107 Das finale DataFrame 'final_df' könnte wie folgt aussehen:
108
109 '''
110
111         Oberbezirk      Unterbezirk      Raub
112 0             Mitte           Unter1      884

```

```

101 1 Friedrichshain-Kreuzberg    Unter2    521
102 2                        Pankow    Unter3    ...
103 ...
104 '''
105
106 ### Hinweise
107
108 - Stellen Sie sicher, dass die Spaltennamen exakt mit denen in Ihrer Excel-
    Datei übereinstimmen. Falls sie sich unterscheiden, passen Sie die
    Spaltennamen im Skript entsprechend an.
109
110 - Falls die Excel-Datei große Datenmengen enthält oder zusätzliche
    Datenbereinigungen notwendig sind, könnten weitere Anpassungen am Skript
    erforderlich sein.
111
112 - Um das Ergebnis in eine neue Excel-Datei zu speichern, können Sie am Ende
    des Skripts folgende Zeile hinzufügen:
113
114     '''python
115     final_df.to_excel('Unterbezirke_mit_maximalem_Raub.xlsx', index=False)
116     '''
117
118 Dieses Skript sollte Ihre Anforderungen erfüllen und Ihnen eine Übersicht der
    Unterbezirke mit den höchsten Raubfällen pro Oberbezirk geben.

```

Listing 174: API Response Testfall 5 Prompt 3 Ausführung 2

```

1  '''python
2  import pandas as pd
3
4  # Schritt 1: Die Daten des Sheets 'Fallzahlen_2023' auslesen
5  excel_file = 'Fallzahlen&HZ 2014-2023.xlsx'
6  sheet_name = 'Fallzahlen_2023'
7
8  # Schritt 2: Die Daten in einem Pandas DataFrame speichern
9  df = pd.read_excel(excel_file, sheet_name=sheet_name)
10
11 # Schritt 3: Das DataFrame auf die notwendigen Spalten reduzieren
12 necessary_columns = ['Bezeichnung (Bezirksregion)', 'Raub']
13 df = df[necessary_columns]
14
15 # Schritt 4: Unnötige Zeilen entfernen (letzte zwei Zeilen)
16 df = df.iloc[:-2].reset_index(drop=True)
17
18 # Schritt 5: Die Oberbezirke definieren
19 oberbezirke = [
20     'Mitte', 'Friedrichshain-Kreuzberg', 'Pankow',
21     'Charlottenburg-Wilmersdorf', 'Spandau', 'Steglitz-Zehlendorf',
22     'Tempelhof-Schöneberg', 'Neukölln', 'Treptow-Köpenick',
23     'Marzahn-Hellersdorf', 'Lichtenberg', 'Reinickendorf'
24 ]
25
26 # Hinzufügen einer neuen Spalte zur Identifizierung des aktuellen Oberbezirks

```

```

27 df['Oberbezirk'] = None
28 current_oberbezirk = None
29
30 for index, row in df.iterrows():
31     bezirk = row['Bezeichnung (Bezirksregion)']
32     if bezirk in oberbezirke:
33         current_oberbezirk = bezirk
34         df.at[index, 'Oberbezirk'] = current_oberbezirk
35     else:
36         df.at[index, 'Oberbezirk'] = current_oberbezirk
37
38 # Schritt 6: Unterbezirke identifizieren (Zeilen, die keine Oberbezirke sind)
39 unterbezirke_df = df[~df['Bezeichnung (Bezirksregion)'].isin(oberbezirke)].
    copy()
40
41 # Sicherstellen, dass 'Raub' als numerischer Wert vorliegt
42 unterbezirke_df['Raub'] = pd.to_numeric(unterbezirke_df['Raub'], errors='
    coerce')
43
44 # Schritt 6: Für jeden Oberbezirk den Unterbezirk mit dem höchsten 'Raub'
    finden
45 max_raub_df = unterbezirke_df.loc[unterbezirke_df.groupby('Oberbezirk')['Raub'
    ].idxmax()]
46
47 # Schritt 7: Die Ergebnisse in einem neuen DataFrame speichern
48 final_df = max_raub_df[['Oberbezirk', 'Bezeichnung (Bezirksregion)', 'Raub'
    ]].rename(
49     columns={
50         'Bezeichnung (Bezirksregion)': 'Unterbezirk',
51         'Raub': 'Raub'
52     }
53 ).reset_index(drop=True)
54
55 # Schritt 8: Das finale DataFrame ausgeben
56 print(final_df)
57 '''
58
59 ### Erklärung der Schritte:
60
61 1. **Daten einlesen:**
62     - Verwenden Sie 'pd.read_excel' um das spezifische Sheet 'Fallzahlen_2023
    ' aus der Excel-Datei zu laden.
63
64 2. **Spalten reduzieren:**
65     - Behalten Sie nur die Spalten 'Bezeichnung (Bezirksregion)' und 'Raub'
    für die weitere Verarbeitung.
66
67 3. **Unnötige Zeilen entfernen:**
68     - Entfernen Sie die letzten zwei Zeilen des DataFrames, da diese
    Gesamtwerte enthalten.
69
70 4. **Oberbezirke definieren und zuweisen:**

```

```

71 - Definieren Sie eine Liste 'oberbezirke' mit den Namen der Oberbezirke.
72 - Iterieren Sie über jede Zeile und weisen Sie die aktuelle Oberbezirk zu.
    Wenn eine Zeile ein Oberbezirk ist, aktualisieren Sie 'current_oberbezirk'
    '. Andernfalls wird die Zeile einem bestehenden Oberbezirk zugewiesen.
73
74 5. **Unterbezirke filtern:**
75 - Filtern Sie das DataFrame, um nur die Unterbezirke zu behalten (d.h.,
    Zeilen, die keine Oberbezirke sind).
76
77 6. **Raub-Werte bereinigen:**
78 - Stellen Sie sicher, dass die Werte in der Spalte 'Raub' numerisch sind
    , um eine korrekte Aggregation zu ermöglichen.
79
80 7. **Maximale Raub-Werte finden:**
81 - Gruppieren Sie die Daten nach 'Oberbezirk' und finden Sie für jede
    Gruppe den Unterbezirk mit dem höchsten 'Raub'-Wert.
82
83 8. **Ergebnisse formatieren:**
84 - Speichern Sie die Ergebnisse in einem neuen DataFrame 'final_df' mit den
    Spalten 'Oberbezirk', 'Unterbezirk' und 'Raub'.
85
86 9. **Ausgabe:**
87 - Drucken Sie das finale DataFrame, das die gewünschten Informationen enth
    ält.
88
89 ### Beispielhafte Ausgabe:
90
91 '''
92         Oberbezirk           Unterbezirk      Raub
93 0             Mitte                Mitte    884
94 1 Friedrichshain-Kreuzberg  Tiergarten Süd      47
95 2             Pankow           Regierungsviertel    44
96 3 Charlottenburg-Wilmersdorf  Alexanderplatz    199
97 4             Spandau      Brunnenstraße Süd     51
98 5      Steglitz-Zehlendorf           MV Nord     29
99 6      Tempelhof-Schöneberg           MV Süd     10
100 7             Neukölln      Bezirk (Rd), nicht zuzuordnen      8
101 8      Treptow-Köpenick      Stadtgebiet Berlin, nicht zuzuordnen    98
102 9      Marzahn-Hellersdorf      Berlin (PKS gesamt)    5389
103 10            Lichtenberg      Berlin (PKS gesamt)    5389
104 11      Reinickendorf      Berlin (PKS gesamt)    5389
105 '''
106
107 *Bitte beachten Sie, dass die tatsächlichen Werte von Ihren spezifischen
    Daten abweichen können.*
108
109 ### Zusätzliche Hinweise:
110
111 - **Fehlerbehandlung:** Stellen Sie sicher, dass die Excel-Datei und das
    Sheet korrekt benannt sind und sich im gleichen Verzeichnis wie das Skript
    befinden oder geben Sie den vollständigen Pfad an.
112

```

```

113 - **Abhängigkeiten:** Dieses Skript erfordert die 'pandas' Bibliothek. Sie kö
      nnen sie installieren, falls noch nicht geschehen, mittels:
114     '''bash
115     pip install pandas
116     '''
117
118 - **Performance:** Für große Datensätze ist die obige Methode effizient. Bei
      sehr großen Excel-Dateien könnten zusätzliche Optimierungen notwendig sein
119     .
120
121 - **Anpassungen:** Falls sich die Struktur der Excel-Datei ändert (z.B. zusä
      tzliche Zeilen oder Spalten), passen Sie die Indizes und Spaltennamen
      entsprechend an.

```

Listing 175: API Response Testfall 5 Prompt 3 Ausführung 3

```

1 Natürlich! Hier ist ein vollständiges Python-Skript, das die von dir
      beschriebenen Schritte durchführt. Das Skript verwendet die 'pandas'-
      Bibliothek, um die Excel-Datei zu verarbeiten und die gewünschten
      Ergebnisse zu erzielen.
2
3 **Voraussetzungen:**
4 - Stelle sicher, dass du die 'pandas'- und 'openpyxl'-Bibliotheken
      installiert hast. Falls nicht, kannst du sie mit 'pip' installieren:
5     '''bash
6     pip install pandas openpyxl
7     '''
8
9 **Python-Skript:**
10
11 '''python
12 import pandas as pd
13
14 # Schritt 1: Excel-Datei lesen und das Sheet 'Fallzahlen_2023' laden
15 datei_name = 'Fallzahlen&HZ 2014-2023.xlsx'
16 sheet_name = 'Fallzahlen_2023'
17
18 # Lese die Excel-Datei
19 try:
20     df = pd.read_excel(datei_name, sheet_name=sheet_name)
21     print(f"Sheet '{sheet_name}' erfolgreich geladen.")
22 except FileNotFoundError:
23     print(f"Die Datei '{datei_name}' wurde nicht gefunden.")
24     exit(1)
25 except ValueError:
26     print(f"Das Sheet '{sheet_name}' existiert nicht in der Datei '{
      datei_name}'")
27     exit(1)
28
29 # Schritt 2: Daten in einem Pandas DataFrame speichern
30 # (Bereits durch pd.read_excel erledigt)
31
32 # Schritt 3: DataFrame auf notwendige Spalten reduzieren

```



```

33 notwendige_spalten = ['Bezeichnung (Bezirksregion)', 'Raub']
34 # Überprüfe, ob alle notwendigen Spalten vorhanden sind
35 fehlende_spalten = [spalte for spalte in notwendige_spalten if spalte not in
    df.columns]
36 if fehlende_spalten:
37     print(f"Die folgenden benötigten Spalten fehlen im DataFrame: {
    fehlende_spalten}")
38     exit(1)
39
40 df = df[notwendige_spalten]
41 print("DataFrame auf notwendige Spalten reduziert.")
42
43 # Schritt 4: Letzte zwei Zeilen entfernen (Gesamtwerte)
44 df = df.iloc[:-2].reset_index(drop=True)
45 print("Letzte zwei Zeilen (Gesamtwerte) entfernt.")
46
47 # Schritt 5: Oberbezirke identifizieren
48 oberbezirke = [
49     'Mitte',
50     'Friedrichshain-Kreuzberg',
51     'Pankow',
52     'Charlottenburg-Wilmersdorf',
53     'Spandau',
54     'Steglitz-Zehlendorf',
55     'Tempelhof-Schöneberg',
56     'Neukölln',
57     'Treptow-Köpenick',
58     'Marzahn-Hellersdorf',
59     'Lichtenberg',
60     'Reinickendorf'
61 ]
62
63 # Füge eine neue Spalte 'Oberbezirk' hinzu, um jedem Unterbezirk den zugehö
    rigen Oberbezirk zuzuweisen
64 df['Oberbezirk'] = None
65 current_oberbezirk = None
66
67 for idx, row in df.iterrows():
68     bezirk = row['Bezeichnung (Bezirksregion)']
69     if bezirk in oberbezirke:
70         current_oberbezirk = bezirk
71         df.at[idx, 'Oberbezirk'] = current_oberbezirk
72     else:
73         df.at[idx, 'Oberbezirk'] = current_oberbezirk
74
75 print("Oberbezirke zugewiesen.")
76
77 # Schritt 6: Oberbezirke von Unterbezirken trennen
78 # Annahme: Oberbezirke selbst sind keine Unterbezirke und sollten nicht berü
    cksichtigt werden
79 df_unterbezirke = df[~df['Bezeichnung (Bezirksregion)'].isin(oberbezirke)].
    copy()

```

```

80 print("Unterbezirke von Oberbezirken getrennt.")
81
82 # Überprüfe, ob jedem Unterbezirk ein Oberbezirk zugewiesen wurde
83 if df_unterbezirke['Oberbezirk'].isnull().any():
84     print("Warnung: Einige Unterbezirke haben keinen zugewiesenen Oberbezirk.")
85
86 # Schritt 7: Für jeden Oberbezirk den Unterbezirk mit dem höchsten 'Raub'
    identifizieren
87 # Zuerst stelle sicher, dass die 'Raub'-Spalte numerisch ist
88 df_unterbezirke['Raub'] = pd.to_numeric(df_unterbezirke['Raub'], errors='
    coerce')
89
90 # Entferne Zeilen mit fehlenden 'Raub'-Werten
91 df_unterbezirke = df_unterbezirke.dropna(subset=['Raub'])
92
93 # Gruppiere nach 'Oberbezirk' und finde den Unterbezirk mit dem maximalen '
    Raub'
94 max_raub_unterbezirke = df_unterbezirke.loc[df_unterbezirke.groupby('
    Oberbezirk')['Raub'].idxmax()].reset_index(drop=True)
95
96 # Schritt 8: Ergebnis in einem neuen DataFrame speichern und formatieren
97 final_df = max_raub_unterbezirke[['Oberbezirk', 'Bezeichnung (Bezirksregion)',
    'Raub']].rename(
98     columns={
99         'Bezeichnung (Bezirksregion)': 'Unterbezirk',
100         'Raub': 'Raub'
101     }
102 )
103
104 # Optional: Sortiere das finale DataFrame nach 'Oberbezirk' für bessere Ü
    bersichtlichkeit
105 final_df = final_df.sort_values(by='Oberbezirk').reset_index(drop=True)
106
107 # Schritt 9: Finale Ergebnisse anzeigen
108 print("\nUnterbezirke mit den höchsten 'Raub'-Werten je Oberbezirk:")
109 print(final_df)
110
111 # Optional: Speichere das Ergebnis in eine neue Excel-Datei
112 output_datei = 'Max_Raub_Unterbezirke_2023.xlsx'
113 final_df.to_excel(output_datei, index=False)
114 print(f"\nErgebnisse wurden in '{output_datei}' gespeichert.")
115 '''
116
117 **Erklärung der Schritte:**
118
119 1. **Excel-Datei lesen:** Das Skript liest die angegebene Excel-Datei und das
    spezifische Sheet 'Fallzahlen_2023'. Fehlerbehandlung ist integriert, um
    sicherzustellen, dass die Datei und das Sheet existieren.
120
121 2. **DataFrame erstellen und reduzieren:** Es werden nur die benötigten
    Spalten 'Bezeichnung (Bezirksregion)' und 'Raub' ausgewählt. Das

```

```

122     Skript überprüft auch, ob diese Spalten tatsächlich vorhanden sind.
123 3. **Unnötige Zeilen entfernen:** Die letzten zwei Zeilen, die Gesamtwerte
124     enthalten, werden entfernt.
125 4. **Oberbezirke identifizieren und zuweisen:** Eine neue Spalte 'Oberbezirk'
126     wird erstellt, in der jeder Unterbezirk den entsprechenden Oberbezirk
127     zugewiesen bekommt. Dies geschieht durch Iteration über die DataFrame-
128     Zeilen.
129 5. **Unterbezirke filtern:** Die Oberbezirke selbst werden aus dem DataFrame
130     entfernt, sodass nur die Unterbezirke verbleiben.
131 6. **Maximale 'Raub'-Werte ermitteln:** Für jeden Oberbezirk wird der
132     Unterbezirk mit dem höchsten Wert in der Spalte 'Raub' identifiziert.
133 7. **Ergebnisse speichern und anzeigen:** Die Ergebnisse werden in einem
134     neuen DataFrame gespeichert, der die Spalten 'Oberbezirk', 'Unterbezirk'
135     und 'Raub' enthält. Dieses DataFrame wird angezeigt und optional in
136     eine neue Excel-Datei gespeichert.
137 **Beispielausgabe:**
138 '''
139 Unterbezirke mit den höchsten 'Raub'-Werten je Oberbezirk:
140
141     Oberbezirk      Unterbezirk      Raub
142 0           Mitte           XYZ           500
143 1  Friedrichshain-Kreuzberg      ABC           450
144 2           Pankow           DEF           600
145 ...           ...           ...           ...
146 '''
147 **Hinweise:**
148 - **Datenüberprüfung:** Es ist wichtig sicherzustellen, dass die Spalte 'Raub'
149     numerische Werte enthält. Das Skript konvertiert diese Spalte in
150     numerische Werte und entfernt Zeilen mit ungültigen Einträgen.
151 - **Fehlende Oberbezirke:** Falls einige Unterbezirke keinen zugewiesenen
152     Oberbezirk haben, gibt das Skript eine Warnung aus. Dies könnte auf
153     Inkonsistenzen in den Daten hinweisen.
154 - **Speicheroption:** Das finale DataFrame wird zusätzlich in eine neue Excel
155     -Datei namens 'Max_Raub_Unterbezirke_2023.xlsx' gespeichert. Du kannst
156     diesen Schritt anpassen oder entfernen, falls nicht benötigt.
157
158 Dieses Skript sollte die gewünschten Schritte effizient ausführen und dir die
159 Unterbezirke mit den höchsten 'Raub'-Werten je Oberbezirk liefern.
160
161 Passe die Dateinamen und Pfade bei Bedarf an deine spezifische Umgebung an
162 .

```

Listing 176: API Response Testfall 5 Prompt 3 Ausführung 4

```

1 Hier ist ein vollständiges Python-Skript, das die von dir beschriebenen
  Schritte ausführt. Das Skript verwendet die Bibliothek 'pandas' zum
  Verarbeiten der Excel-Daten. Stelle sicher, dass du die benötigten
  Bibliotheken installiert hast ('pandas' und 'openpyxl'), bevor du das
  Skript ausführst.

2
3 '''python
4 import pandas as pd
5
6 # Schritt 1: Excel-Datei und das spezifische Sheet laden
7 excel_file = 'Fallzahlen&HZ 2014-2023.xlsx'
8 sheet_name = 'Fallzahlen_2023'
9
10 # Schritt 2: Daten in einem Pandas DataFrame speichern
11 df = pd.read_excel(excel_file, sheet_name=sheet_name)
12
13 # Schritt 3: DataFrame auf die notwendigen Spalten reduzieren
14 selected_columns = ['Bezeichnung (Bezirksregion)', 'Raub']
15 df = df[selected_columns]
16
17 # Schritt 4: Die letzten zwei Zeilen entfernen (Gesamtwerte)
18 df = df.iloc[:-2].reset_index(drop=True)
19
20 # Schritt 5: Liste der Oberbezirke definieren
21 oberbezirke = [
22     'Mitte',
23     'Friedrichshain-Kreuzberg',
24     'Pankow',
25     'Charlottenburg-Wilmersdorf',
26     'Spandau',
27     'Steglitz-Zehlendorf',
28     'Tempelhof-Schöneberg',
29     'Neukölln',
30     'Treptow-Köpenick',
31     'Marzahn-Hellersdorf',
32     'Lichtenberg',
33     'Reinickendorf'
34 ]
35
36 # Initialisierung
37 results = []
38 current_oberbezirk = None
39
40 # Schritt 6: Durch das DataFrame iterieren und Unterbezirke zuordnen
41 for index, row in df.iterrows():
42     bezirk = row['Bezeichnung (Bezirksregion)']
43     raub = row['Raub']
44
45     if bezirk in oberbezirke:
46         # Aktuellen Oberbezirk setzen
47         current_oberbezirk = bezirk
48     else:

```

```

49         if current_oberbezirk:
50             # Unterbezirk und Raub-Wert hinzufügen
51             results.append({
52                 'Oberbezirk': current_oberbezirk,
53                 'Unterbezirk': bezirk,
54                 'Raub': raub
55             })
56
57 # In einen neuen DataFrame umwandeln
58 results_df = pd.DataFrame(results)
59
60 # Schritt 6 (Fortsetzung): Für jeden Oberbezirk den Unterbezirk mit dem hö
    chsten 'Raub' finden
61 max_raub_df = results_df.loc[results_df.groupby('Oberbezirk')['Raub'].idxmax
    ()].reset_index(drop=True)
62
63 # Schritt 7: Spalten umbenennen (falls nötig) und finalen DataFrame erstellen
64 final_df = max_raub_df[['Oberbezirk', 'Unterbezirk', 'Raub']]
65
66 # Schritt 8: Finalen DataFrame ausgeben
67 print(final_df)
68 '''
69
70 ### Erklärung der Schritte:
71
72 1. **Daten laden**: Das Skript lädt die Excel-Datei 'Fallzahlen&HZ 2014-2023.
    xlsx' und das spezifische Sheet 'Fallzahlen_2023' mithilfe von 'pd.
    read_excel'.
73
74 2. **Spalten auswählen**: Es werden nur die Spalten 'Bezeichnung (
    Bezirksregion)' und 'Raub' ausgewählt, da diese für die Analyse benö
    tigt werden.
75
76 3. **Unnötige Zeilen entfernen**: Die letzten zwei Zeilen des DataFrames, die
    Gesamtwerte enthalten, werden entfernt.
77
78 4. **Oberbezirke definieren**: Eine Liste der Oberbezirke wird erstellt, um
    diese später zu identifizieren.
79
80 5. **Zuordnung von Unterbezirken zu Oberbezirken**: Das Skript iteriert durch
    jede Zeile des DataFrames. Wenn ein Oberbezirk gefunden wird, wird er als
    aktueller Oberbezirk festgelegt. Alle folgenden Zeilen bis zum nächsten
    Oberbezirk werden als Unterbezirke dieses Oberbezirks betrachtet.
81
82 6. **Identifikation der Unterbezirke mit dem höchsten Raub**: Für jeden
    Oberbezirk wird der Unterbezirk mit dem höchsten Wert in der Spalte 'Raub
    ' identifiziert.
83
84 7. **Erstellen des finalen DataFrames**: Die Ergebnisse werden in einem neuen
    DataFrame 'final_df' gespeichert, welches die Spalten 'Oberbezirk', '
    Unterbezirk' und 'Raub' enthält.
85

```

```

86 8. **Ausgabe**: Der finale DataFrame wird ausgegeben, sodass du die gewü
    nschten Informationen sehen kannst.
87
88 ### Zusätzliche Hinweise:
89
90 - **Bibliotheken installieren**: Stelle sicher, dass du 'pandas' und '
    openpyxl' installiert hast. Falls nicht, kannst du sie mit folgendem
    Befehl installieren:
91
92     '''bash
93     pip install pandas openpyxl
94     '''
95
96 - **Dateipfad anpassen**: Stelle sicher, dass sich die Excel-Datei im
    gleichen Verzeichnis wie das Skript befindet oder passe den 'excel_file'
    Pfad entsprechend an.
97
98 - **Überprüfung der Spaltennamen**: Achte darauf, dass die Spaltennamen in
    der Excel-Datei exakt mit den im Skript verwendeten Namen übereinstimmen.
    Andernfalls passe die Namen im Skript entsprechend an.
99
100 Dieses Skript sollte dir die gewünschten Ergebnisse liefern, indem es die
    Unterbezirke mit den höchsten Raubfällen für jeden Oberbezirk
    identifiziert und in einem übersichtlichen DataFrame darstellt.

```

Listing 177: API Response Testfall 5 Prompt 3 Ausführung 5

11.3.13 Komplette Antworten der API Requests Testfall 6

```

1  Gerne helfe ich dir dabei, ein Python-Skript zu erstellen, das die gewü
    nschten Daten aus deiner Excel-Datei verarbeitet. Wir werden die
    Bibliothek 'pandas' verwenden, die hervorragend für Datenanalyse-Aufgaben
    geeignet ist.
2
3  ### Voraussetzungen
4
5  1. **Python installieren**: Stelle sicher, dass Python auf deinem Computer
    installiert ist. Du kannst es von [python.org](https://www.python.org/
    downloads/) herunterladen.
6
7  2. **Benötigte Bibliotheken installieren**: Öffne dein Terminal oder die
    Eingabeaufforderung und installiere die notwendigen Bibliotheken mit 'pip
    ':
8
9     '''bash
10     pip install pandas openpyxl
11     '''
12
13     - 'pandas': Für die Datenverarbeitung.
14     - 'openpyxl': Zum Lesen von Excel-Dateien.
15
16 ### Skript erstellen

```

```

17
18 Erstelle eine neue Python-Datei, z.B. 'top_unterbezirke.py', und füge
    folgenden Code ein:
19
20 '''python
21 import pandas as pd
22
23 def get_top_unterbezirke(excel_file, sheets_prefix='Fallzahlen_', start_year
    =2014, end_year=2023, top_n=10):
24     # Liste zur Speicherung der DataFrames aus allen Jahren
25     df_list = []
26
27     for year in range(start_year, end_year + 1):
28         sheet_name = f"{sheets_prefix}{year}"
29         try:
30             # Lade das jeweilige Sheet
31             df = pd.read_excel(excel_file, sheet_name=sheet_name)
32
33             # Überprüfe, ob die notwendigen Spalten vorhanden sind
34             if 'LOR-Schlüssel (Bezirksregion)' in df.columns and 'Straftaten
    insgesamt' in df.columns:
35                 # Wähle relevante Spalten
36                 df_subset = df[['LOR-Schlüssel (Bezirksregion)', 'Bezeichnung
    (Bezirksregion)', 'Straftaten insgesamt']].copy()
37
38                 # Falls es weitere Bereinigungen benötigt, kann dies hier
    eingefügt werden
39
40                 df_list.append(df_subset)
41             else:
42                 print(f"Wichtige Spalten fehlen im Sheet {sheet_name}.")
43         except Exception as e:
44             print(f"Fehler beim Laden des Sheets {sheet_name}: {e}")
45
46         if not df_list:
47             print("Keine Daten zum Verarbeiten gefunden.")
48             return
49
50     # Kombiniere alle DataFrames
51     combined_df = pd.concat(df_list, ignore_index=True)
52
53     # Gruppiere nach Unterbezirk und summiere die Straftaten insgesamt
54     aggregated_df = combined_df.groupby(['LOR-Schlüssel (Bezirksregion)', '
    Bezeichnung (Bezirksregion)'])['Straftaten insgesamt'].sum().reset_index()
55
56     # Sortiere nach den meisten Straftaten und wähle die Top N
57     top_unterbezirke = aggregated_df.sort_values(by='Straftaten insgesamt',
    ascending=False).head(top_n)
58
59     return top_unterbezirke
60
61 if __name__ == "__main__":

```

```

62     excel_file = 'Fallzahlen&HZ 2014-2023.xlsx' # Pfad zu deiner Excel-Datei
63     top_n = 10 # Anzahl der Top-Unterbezirke
64
65     top_unterbezirke = get_top_unterbezirke(excel_file, top_n=top_n)
66
67     if top_unterbezirke is not None:
68         print(f"Top {top_n} Unterbezirke mit den meisten Straftaten insgesamt
69             (2014-2023):")
70         print(top_unterbezirke.to_string(index=False))
71     '''
72
73     ### Erklärung des Skripts
74
75     1. **Funktion 'get_top_unterbezirke'**:
76         - **Parameter**:
77             - 'excel_file': Pfad zur Excel-Datei.
78             - 'sheets_prefix': Prefix der Sheet-Namen (standardmäßig '
79             Fallzahlen_').
80             - 'start_year' und 'end_year': Jahrsspanne (standardmäßig 2014 bis
81             2023).
82             - 'top_n': Anzahl der Top-Unterbezirke, die zurückgegeben werden
83             sollen.
84         - **Ablauf**:
85             - Iteriert über die angegebenen Jahre und lädt jedes entsprechende
86             Sheet.
87             - Überprüft, ob die notwendigen Spalten ('LOR-Schlüssel (
88             Bezirksregion)' und 'Straftaten insgesamt') vorhanden sind.
89             - Extrahiert die relevanten Spalten und fügt sie einer Liste hinzu.
90             - Nach dem Laden aller Sheets kombiniert den DataFrame und gruppiert
91             die Daten nach Unterbezirk, wobei die Straftaten summiert werden.
92             - Sortiert die Ergebnisse nach der Anzahl der Straftaten und gibt die
93             Top N zurück.
94
95     2. **Hauptteil des Skripts**:
96         - Definiert den Pfad zur Excel-Datei und die gewünschte Anzahl der Top-
97         Unterbezirke.
98         - Ruft die Funktion auf und gibt das Ergebnis aus.
99
100     ### Skript ausführen
101
102     Speichere die Datei 'top_unterbezirke.py' und führe sie über das Terminal
103     oder die Eingabeaufforderung aus:
104
105     '''bash
106     python top_unterbezirke.py
107     '''
108
109     ### Beispielausgabe
110
111     '''
112     Top 10 Unterbezirke mit den meisten Straftaten insgesamt (2014-2023):

```



```

103 LOR-Schlüssel (Bezirksregion) Bezeichnung (Bezirksregion) Straftaten
    insgesamt
104          999999          Berlin (PKS gesamt)
    XXXXXXX
105          10000          Mitte          XXXXXXX
106          11003          Alexanderplatz          XXXXXXX
107          ...          ...          ...
108 '''
109
110 *(Hinweis: 'XXXXXXX' steht hier für die tatsächlichen summierten Straftaten.)
    *
111
112 ### Anpassungen und Erweiterungen
113
114 - **Weitere Analysen**: Das Skript kann erweitert werden, um detailliertere
    Analysen durchzuführen, z.B. nach spezifischen Straftatenarten.
115 - **Visualisierung**: Mit Bibliotheken wie 'matplotlib' oder 'seaborn' kannst
    du die Ergebnisse visualisieren.
116 - **Fehlerbehandlung**: Das Skript enthält grundlegende Fehlerbehandlungen.
    Je nach Struktur deiner Daten könnten weitere Anpassungen notwendig sein.
117
118 Ich hoffe, dieses Skript hilft dir weiter! Wenn du weitere Fragen hast oder
    Unterstützung benötigst, stehe ich gerne zur Verfügung.

```

Listing 178: API Response Testfall 6 Prompt 1 Ausführung 1

```

1 Gerne helfe ich Ihnen dabei, ein Python-Skript zu erstellen, das Ihre Excel-
    Datei verarbeitet und die gewünschten Informationen extrahiert. Wir werden
    die Bibliotheken 'pandas' und 'openpyxl' verwenden, um mit Excel-Dateien
    zu arbeiten. Falls Sie diese Bibliotheken noch nicht installiert haben, kö
    nnen Sie dies mit 'pip' tun:
2
3 '''bash
4 pip install pandas openpyxl
5 '''
6
7 Hier ist ein vollständiges Skript, das Ihre Anforderungen erfüllt:
8
9 '''python
10 import pandas as pd
11
12 def get_top_10_unterbezirke(excel_file):
13     # Liste der Jahre von 2014 bis 2023
14     jahre = list(range(2014, 2024))
15
16     # Initialisiere ein leeres DataFrame für die aggregierten Daten
17     aggregierte_daten = pd.DataFrame()
18
19     # Iteriere über jede Jahres-Sheet
20     for jahr in jahre:
21         sheet_name = f'Fallzahlen_{jahr}'
22         try:
23             # Lese das aktuelle Sheet

```

```

24         df = pd.read_excel(excel_file, sheet_name=sheet_name)
25
26         # Überprüfe, ob die notwendigen Spalten vorhanden sind
27         if 'Bezeichnung (Bezirksregion)' not in df.columns or 'Straftaten
insgesamt' not in df.columns:
28             print(f"Wichtige Spalten fehlen im Sheet {sheet_name}. Ü
berspringe dieses Sheet.")
29             continue
30
31         # Auswahl der relevanten Spalten
32         # Annahme: 'Straftaten insgesamt' ist die Spalte mit den gesamten
Straftaten
33         # Passen Sie die Spaltennamen an, falls sie sich unterscheiden
34         df_relevant = df[['Bezeichnung (Bezirksregion)', 'Straftaten
insgesamt']]
35
36         # Gruppieren nach Unterbezirk und summiere die Straftaten
37         df_grouped = df_relevant.groupby('Bezeichnung (Bezirksregion)',
as_index=False).sum()
38
39         # Füge die Daten dem aggregierten DataFrame hinzu
40         if aggregierte_daten.empty:
41             aggregierte_daten = df_grouped
42         else:
43             aggregierte_daten = aggregierte_daten.merge(df_grouped, on='
Bezeichnung (Bezirksregion)', how='outer')
44             aggregierte_daten['Straftaten insgesamt'] = aggregierte_daten
['Straftaten insgesamt_x'].fillna(0) + aggregierte_daten['Straftaten
insgesamt_y'].fillna(0)
45             aggregierte_daten.drop(['Straftaten insgesamt_x', 'Straftaten
insgesamt_y'], axis=1, inplace=True)
46
47         except ValueError:
48             print(f"Sheet {sheet_name} nicht gefunden in der Datei. Ü
berspringe dieses Sheet.")
49             continue
50
51         # Nach allen Sheets aggregiert, sortiere die Unterbezirke nach Straftaten
insgesamt
52         aggregierte_daten_sorted = aggregierte_daten.sort_values(by='Straftaten
insgesamt', ascending=False)
53
54         # Wähle die Top 10 Unterbezirke aus
55         top_10 = aggregierte_daten_sorted.head(10)
56
57         return top_10
58
59 if __name__ == "__main__":
60     excel_datei = 'Fallzahlen&HZ 2014-2023.xlsx'
61     top_10_unterbezirke = get_top_10_unterbezirke(excel_datei)
62     print("Die Top 10 Unterbezirke mit den meisten Straftaten insgesamt
(2014-2023):\n")

```

```

63     print(top_10_unterbezirke.to_string(index=False))
64     '''
65
66     ### Erläuterung des Skripts
67
68     1. **Importieren der Bibliotheken:**
69     - 'pandas' wird verwendet, um Excel-Dateien zu lesen und Daten zu
        verarbeiten.
70
71     2. **Funktion 'get_top_10_unterbezirke':**
72     - **Parameter:** 'excel_file' - der Pfad zu Ihrer Excel-Datei.
73     - **Jahre:** Wir erstellen eine Liste der Jahre von 2014 bis 2023, um
        iterativ durch die entsprechenden Sheets zu gehen.
74     - **Iterieren über die Sheets:**
75         - Für jedes Jahr generieren wir den Sheet-Namen (z.B. 'Fallzahlen_2014')
        .
76         - Wir versuchen, den Sheet zu lesen. Falls der Sheet nicht existiert,
        wird eine Fehlermeldung ausgegeben und das Sheet übersprungen.
77         - **Wichtige Spalten prüfen:** Stellen Sie sicher, dass die Spalten '
        Bezeichnung (Bezirksregion)' und 'Straftaten insgesamt' vorhanden sind.
        Passen Sie die Spaltennamen an, falls diese anders benannt sind.
78         - **Auswahl und Gruppierung:**
79         - Wir wählen die relevanten Spalten aus und gruppieren die Daten nach
        'Bezeichnung (Bezirksregion)', wobei wir die Straftaten summieren.
80         - **Aggregation über Jahre hinweg:**
81         - Wenn das 'aggregierte_daten' DataFrame leer ist, fügen wir die
        gruppierten Daten hinzu.
82         - Andernfalls fügen wir die neuen Daten hinzu und summieren die '
        Straftaten insgesamt' aus den verschiedenen Jahren.
83
84     3. **Sortieren und Auswählen der Top 10:**
85     - Nachdem alle Jahre verarbeitet wurden, sortieren wir die Unterbezirke
        nach der Gesamtzahl der Straftaten in absteigender Reihenfolge.
86     - Wir wählen die Top 10 Einträge aus.
87
88     4. **Ausführung des Skripts:**
89     - Das Skript liest die Excel-Datei 'Fallzahlen&HZ 2014-2023.xlsx',
        berechnet die Top 10 Unterbezirke und gibt die Ergebnisse aus.
90
91     ### Anpassungen und Hinweise
92
93     - **Spaltennamen:** Stellen Sie sicher, dass die Spaltennamen in Ihrem Excel-
        Sheet genau mit den im Skript verwendeten Namen übereinstimmen. Falls sie
        abweichen, passen Sie die Namen entsprechend an.
94
95     - **Datenbereinigung:** Falls es fehlende Werte oder Inkonsistenzen in Ihren
        Daten gibt, sollten zusätzliche Datenbereinigungsschritte hinzugefügt
        werden.
96
97     - **Leistung:** Bei sehr großen Excel-Dateien könnte das Skript optimiert
        werden, um die Leistung zu verbessern, z.B. durch das Lesen nur
        spezifischer Spalten.

```

```

98
99 ### Beispielausgabe
100
101 Nach erfolgreichem Ausführen des Skripts erhalten Sie eine Ausgabe ähnlich
    der folgenden:
102
103 '''
104 Die Top 10 Unterbezirke mit den meisten Straftaten insgesamt (2014-2023):
105
106     Bezeichnung (Bezirksregion)   Straftaten insgesamt
107                               Mitte           836110
108                               Tiergarten Süd      44910
109                               Regierungsviertel    65500
110                               Alexanderplatz      196730
111                               Brunnenstraße Süd    37240
112                               ...                 ...
113 '''
114
115 Diese Tabelle zeigt die Top 10 Unterbezirke mit den höchsten Straftaten über
    den Zeitraum von 2014 bis 2023.
116
117 ### Ausführen des Skripts
118
119 1. **Speichern Sie das Skript:** Speichern Sie das oben dargestellte Skript
    in einer Datei, z.B. 'top_unterbezirke.py'.
120
121 2. **Stellen Sie sicher, dass sich die Excel-Datei im selben Verzeichnis
    befindet oder geben Sie den vollständigen Pfad an.**
122
123 3. **Führen Sie das Skript aus:**
124
125     '''bash
126     python top_unterbezirke.py
127     '''
128
129 Das Skript wird die Excel-Datei verarbeiten und die Top 10 Unterbezirke mit
    den meisten Straftaten über alle Jahre hinweg anzeigen.
130
131 ### Erweiterungen
132
133 - **Export der Ergebnisse:** Sie können die Ergebnisse auch in eine neue
    Excel- oder CSV-Datei exportieren, indem Sie folgende Zeile am Ende der
    Funktion hinzufügen:
134
135     '''python
136     top_10.to_csv('Top_10_Unterbezirke_Straftaten.csv', index=False)
137     '''
138
139 - **Interaktive Visualisierung:** Verwenden Sie Bibliotheken wie 'matplotlib'
    oder 'seaborn', um die Ergebnisse zu visualisieren.
140
141 Ich hoffe, dieses Skript hilft Ihnen weiter! Bei weiteren Fragen stehe ich

```

gerne zur Verfügung.

Listing 179: API Response Testfall 6 Prompt 1 Ausführung 2

```
1 Natürlich! Hier ist ein Python-Skript, das die gewünschten Aufgaben erfüllt.  
   Dieses Skript verwendet die 'pandas'-Bibliothek, um die Excel-Datei zu  
   verarbeiten. Stellen Sie sicher, dass Sie 'pandas' und 'openpyxl'  
   installiert haben, bevor Sie das Skript ausführen.  
2  
3 ### Schritt 1: Installation der benötigten Bibliotheken  
4  
5 Stellen Sie sicher, dass Sie die erforderlichen Bibliotheken installiert  
   haben. Sie können dies über 'pip' tun:  
6  
7 '''bash  
8 pip install pandas openpyxl  
9 '''  
10  
11 ### Schritt 2: Python-Skript erstellen  
12  
13 Hier ist das vollständige Python-Skript:  
14  
15 '''python  
16 import pandas as pd  
17  
18 # Pfad zur Excel-Datei  
19 excel_file = 'Fallzahlen&HZ 2014-2023.xlsx'  
20  
21 # Liste der relevanten Sheet-Namen  
22 sheet_names = [f'Fallzahlen_{jahr}' for jahr in range(2014, 2024)]  
23  
24 # Initialisiere ein leeres DataFrame zur Speicherung aller Daten  
25 all_data = pd.DataFrame()  
26  
27 # Iteriere über alle relevanten Sheets und füge die Daten hinzu  
28 for sheet in sheet_names:  
29     try:  
30         # Lese das aktuelle Sheet  
31         df = pd.read_excel(excel_file, sheet_name=sheet)  
32  
33         # Überprüfe, ob die notwendigen Spalten vorhanden sind  
34         if 'Bezeichnung (Bezirksregion)' in df.columns and 'Straftaten \n-  
35 insgesamt-' in df.columns:  
36             # Wähle nur die relevanten Spalten  
37             df_subset = df[['Bezeichnung (Bezirksregion)', 'Straftaten \n-  
38 insgesamt-']]  
39  
40             # Entferne mögliche NaN-Werte und stelle sicher, dass 'Straftaten  
41 \n-insgesamt-' numerisch ist  
42             df_subset = df_subset.dropna(subset=['Straftaten \n-insgesamt-'])  
43             df_subset['Straftaten \n-insgesamt-'] = pd.to_numeric(df_subset['  
44 Straftaten \n-insgesamt-'], errors='coerce').fillna(0)  
45             all_data = all_data.append(df_subset)  
46         except Exception as e:  
47             print(f'Fehler beim Lesen des Sheets {sheet}: {e}')  
48 '''
```

```

42         # Füge die Daten zum all_data DataFrame hinzu
43         all_data = pd.concat([all_data, df_subset], ignore_index=True)
44     else:
45         print(f"Warnung: Die notwendigen Spalten sind im Sheet '{sheet}'
nicht vorhanden.")
46     except Exception as e:
47         print(f"Fehler beim Verarbeiten des Sheets '{sheet}': {e}")
48
49 # Gruppieren die Daten nach Unterbezirk und summiere die Straftaten
50 total_straftaten = all_data.groupby('Bezeichnung (Bezirksregion)')['
Straftaten \n-insgesamt-'].sum()
51
52 # Sortiere die Unterbezirke nach den meisten Straftaten
53 top_10_unterbezirke = total_straftaten.sort_values(ascending=False).head(10)
54
55 # Ausgabe der Ergebnisse
56 print("Die Top 10 Unterbezirke mit den meisten Straftaten (2014-2023):")
57 print(top_10_unterbezirke)
58 '''
59
60 ### Schritt 3: Erklärung des Skripts
61
62 1. **Importieren der Bibliothek:**
63     - Das Skript verwendet 'pandas' zum Lesen und Verarbeiten der Excel-Datei.
64
65 2. **Definieren des Excel-Dateipfads:**
66     - Stellen Sie sicher, dass sich die Datei 'Fallzahlen&HZ 2014-2023.xlsx'
im gleichen Verzeichnis wie das Skript befindet oder geben Sie den vollständigen Pfad an.
67
68 3. **Erstellen der Liste der Sheet-Namen:**
69     - Generiert eine Liste von Sheet-Namen von 'Fallzahlen_2014' bis '
Fallzahlen_2023'.
70
71 4. **Initialisieren eines leeren DataFrames:**
72     - Alle Daten aus den relevanten Sheets werden hier gesammelt.
73
74 5. **Iterieren über alle Sheets:**
75     - Für jedes Jahr wird das entsprechende Sheet gelesen.
76     - Es wird überprüft, ob die notwendigen Spalten ('Bezeichnung (
Bezirksregion)' und 'Straftaten \n-insgesamt-') vorhanden sind.
77     - Die relevanten Spalten werden extrahiert, bereinigt und zum 'all_data'
DataFrame hinzugefügt.
78
79 6. **Gruppieren und Summieren der Straftaten:**
80     - Die Daten werden nach 'Bezeichnung (Bezirksregion)' gruppiert, und die
Straftaten werden summiert.
81
82 7. **Sortieren und Auswahl der Top 10:**
83     - Die Unterbezirke werden nach der Gesamtzahl der Straftaten sortiert, und
die Top 10 werden ausgewählt.
84

```

```

85 8. **Ausgabe der Ergebnisse:**
86   - Die Top 10 Unterbezirke mit den meisten Straftaten von 2014 bis 2023
      werden im Terminal angezeigt.
87
88 ### Schritt 4: Ausführen des Skripts
89
90 Speichern Sie das Skript beispielsweise als 'top_unterbezirke.py' und führen
      Sie es über die Kommandozeile oder eine Python-IDE aus:
91
92 '''bash
93 python top_unterbezirke.py
94 '''
95
96 ### Beispielausgabe
97
98 Die Ausgabe könnte wie folgt aussehen:
99
100 '''
101 Die Top 10 Unterbezirke mit den meisten Straftaten (2014-2023):
102 Bezeichnung (Bezirksregion)
103 Berlin (PKS gesamt)      5366950
104 Mitte                    836110
105 Alexanderplatz          196730
106 ...
107 Name: Straftaten \n-insgesamt-, dtype: int64
108 '''
109
110 Diese Ausgabe zeigt die Unterbezirke und die Gesamtzahl der Straftaten über
      den angegebenen Zeitraum.
111
112 ### Hinweise
113
114 - **Spaltennamen überprüfen:** Stellen Sie sicher, dass die Spaltennamen in
      den Excel-Sheets exakt mit denen im Skript übereinstimmen (einschließlich
      Leerzeichen und Zeilenumbrüchen). Falls die Spaltennamen variieren, passen
      Sie das Skript entsprechend an.
115
116 - **Fehlende Daten:** Das Skript behandelt fehlende oder nicht numerische
      Werte, indem es sie entfernt bzw. mit '0' auffüllt. Überprüfen Sie die
      Daten auf Konsistenz für genauere Ergebnisse.
117
118 - **Leistungsoptimierungen:** Bei sehr großen Excel-Dateien könnte die
      Leistung beeinträchtigt werden. In solchen Fällen können Optimierungen
      oder alternative Methoden zur Datenverarbeitung in Betracht gezogen werden
      .
119
120 Ich hoffe, dieses Skript hilft Ihnen weiter! Wenn Sie weitere Anpassungen ben
      ötigen oder Fragen haben, lassen Sie es mich wissen.

```

Listing 180: API Response Testfall 6 Prompt 1 Ausführung 3

```

1 Gerne stelle ich dir ein Python-Skript zur Verfügung, das mit der Bibliothek
  'pandas' arbeitet, um die gewünschten Daten aus deiner Excel-Datei zu

```

```

    verarbeiten. Das Skript liest alle relevanten Sheets ('Fallzahlen_2014'
    bis 'Fallzahlen_2023'), aggregiert die Gesamtzahl der Straftaten pro
    Unterbezirk über alle Jahre hinweg und gibt die Top 10 Unterbezirke mit
    den meisten Straftaten aus.
2
3 ### Voraussetzungen
4
5 Stelle sicher, dass du Python installiert hast und die Bibliothek 'pandas'
    sowie 'openpyxl' für das Einlesen von Excel-Dateien installiert sind. Du
    kannst sie mit pip installieren:
6
7 '''bash
8 pip install pandas openpyxl
9 '''
10
11 ### Skript: 'top10_unterbezirke.py'
12
13 '''python
14 import pandas as pd
15
16 # Pfad zur Excel-Datei
17 excel_file = 'Fallzahlen&HZ 2014-2023.xlsx'
18
19 # Liste der Jahre, die verarbeitet werden sollen
20 jahre = range(2014, 2024) # 2014 bis 2023
21
22 # Initialisiere ein leeres DataFrame für die Aggregation
23 gesamt_df = pd.DataFrame()
24
25 # Iteriere über jedes Jahr und lese das entsprechende Sheet
26 for jahr in jahre:
27     sheet_name = f'Fallzahlen_{jahr}'
28     try:
29         # Lese das Sheet
30         df = pd.read_excel(excel_file, sheet_name=sheet_name)
31
32         # Überprüfe, ob die notwendigen Spalten vorhanden sind
33         notwendige_spalten = ['Bezeichnung (Bezirksregion)', 'Straftaten
    insgesamt']
34         if not all(spalte in df.columns for spalte in notwendige_spalten):
35             print(f"Warnung: Nicht alle notwendigen Spalten in Sheet {
    sheet_name} gefunden.")
36             continue
37
38         # Wähle nur die relevanten Spalten
39         df_relevant = df[['Bezeichnung (Bezirksregion)', 'Straftaten
    insgesamt']]
40
41         # Benenne die Spalten um, um das Jahr zu identifizieren
42         df_relevant = df_relevant.rename(columns={'Straftaten insgesamt': f'
    Straftaten_{jahr}'})
43

```



```

44     # Setze die Bezirksregion als Index
45     df_relevant.set_index('Bezeichnung (Bezirksregion)', inplace=True)
46
47     # Füge zum Gesamt-DataFrame hinzu
48     if gesamt_df.empty:
49         gesamt_df = df_relevant
50     else:
51         gesamt_df = gesamt_df.join(df_relevant, how='outer')
52
53     except FileNotFoundError:
54         print(f"Fehler: Die Datei {excel_file} wurde nicht gefunden.")
55         exit(1)
56     except ValueError:
57         print(f"Warnung: Das Sheet {sheet_name} wurde in der Datei {
58 excel_file} nicht gefunden.")
59         continue
60
61 # Ersetze fehlende Werte (falls ein Bezirk in einem Jahr fehlt) mit 0
62 gesamt_df.fillna(0, inplace=True)
63
64 # Berechne die Gesamtsumme der Straftaten über alle Jahre hinweg
65 gesamt_df['Straftaten_Gesamt'] = gesamt_df.sum(axis=1)
66
67 # Sortiere die Bezirke nach der Gesamtsumme der Straftaten in absteigender
68 Reihenfolge
69 top10 = gesamt_df['Straftaten_Gesamt'].sort_values(ascending=False).head(10)
70
71 # Optional: Zurück in ein DataFrame für bessere Darstellung
72 top10_df = top10.reset_index()
73 top10_df.columns = ['Bezirksregion', 'Gesamt_Straftaten']
74
75 # Ausgabe der Top 10 Bezirke
76 print("Top 10 Unterbezirke mit den meisten Straftaten (2014-2023):")
77 print(top10_df)
78
79 # Optional: Speichern der Ergebnisse in eine neue Excel-Datei
80 # top10_df.to_excel('Top10_Unterbezirke_Straftaten_2014-2023.xlsx', index=
81 False)
82
83 '''
84
85 ### Erklärung des Skripts
86
87 1. **Import der Bibliothek**: Das Skript verwendet 'pandas' zum Verarbeiten
88 der Excel-Datei.
89
90 2. **Festlegung des Dateipfads und der Jahre**: Der Pfad zur Excel-Datei wird
91 festgelegt, und eine Liste der Jahre von 2014 bis 2023 erstellt.
92
93 3. **Initialisierung eines leeren DataFrames**: Dieses DataFrame wird
94 verwendet, um die aggregierten Daten aus allen Sheets zu speichern.
95
96 4. **Schleife über jedes Jahr**:

```

```

90     - Das entsprechende Sheet ('Fallzahlen_YYYY') wird gelesen.
91     - Es wird geprüft, ob die notwendigen Spalten ('Bezeichnung (
    Bezirksregion)' und 'Straftaten insgesamt') vorhanden sind.
92     - Die relevanten Spalten werden ausgewählt und die Straftaten-Spalte wird
    umbenannt, um das Jahr zu identifizieren.
93     - Die Bezirksregion wird als Index gesetzt, und die Daten werden mit dem
    Gesamt-DataFrame zusammengeführt.
94
95 5. **Fehlende Werte behandeln**: Wenn ein Bezirk in einem Jahr nicht
    vorhanden ist, werden die fehlenden Werte mit 0 aufgefüllt.
96
97 6. **Berechnung der Gesamtsumme**: Für jeden Bezirk wird die Summe der
    Straftaten über alle Jahre berechnet.
98
99 7. **Sortierung und Auswahl der Top 10**: Die Bezirke werden nach der
    Gesamtsumme der Straftaten sortiert, und die Top 10 werden ausgewählt.
100
101 8. **Ausgabe der Ergebnisse**: Die Top 10 Bezirke werden in der Konsole
    ausgegeben. Optional kann das Ergebnis auch in eine neue Excel-Datei
    gespeichert werden (diese Zeile ist auskommentiert und kann bei Bedarf
    aktiviert werden).
102
103 ### Ausführen des Skripts
104
105 1. Speichere das oben stehende Skript in einer Datei, z.B. '
    top10_unterbezirke.py'.
106
107 2. Stelle sicher, dass sich die Excel-Datei 'Fallzahlen&HZ 2014-2023.xlsx' im
    gleichen Verzeichnis wie das Skript befindet oder passe den 'excel_file'-
    Pfad entsprechend an.
108
109 3. Führe das Skript über die Kommandozeile aus:
110
111 '''bash
112 python top10_unterbezirke.py
113 '''
114
115 ### Beispielausgabe
116
117 Nach dem Ausführen des Skripts erhältst du eine Ausgabe ähnlich der folgenden
    :
118
119 '''
120 Top 10 Unterbezirke mit den meisten Straftaten (2014-2023):
121 Bezirksregion    Gesamt_Straftaten
122 0                Mitte                83611.0
123 1                Tiergarten Süd        4491.0
124 2                Regierungsviertel      6550.0
125 3                Alexanderplatz        19673.0
126 4                Brunnenstraße Süd      3724.0
127 5                ...                    ...
128 '''

```

```

129
130 Die tatsächlichen Werte hängen von den Daten in deiner Excel-Datei ab.
131
132 ---
133
134 Falls du weitere Anpassungen benötigst oder Fragen hast, stehe ich gerne zur
    Verfügung!

```

Listing 181: API Response Testfall 6 Prompt 1 Ausführung 4

```

1 Natürlich, hier ist ein Python-Skript, das die gewünschten Anforderungen erfü
  llt. Das Skript verwendet die 'pandas' Bibliothek, um die Excel-Datei zu
  verarbeiten und die Top 10 Unterbezirke nach Gesamtzahl der Straftaten zu
  ermitteln.
2
3 ### Voraussetzungen
4
5 1. **Python installieren**: Stellen Sie sicher, dass Python auf Ihrem
  Computer installiert ist. Sie können es von [python.org](https://www.
  python.org/downloads/) herunterladen und installieren.
6
7 2. **Benötigte Bibliotheken installieren**: Öffnen Sie ein Terminal oder eine
  Eingabeaufforderung und installieren Sie die notwendigen Python-
  Bibliotheken mit 'pip':
8
9     '''bash
10     pip install pandas openpyxl
11     '''
12
13     - 'pandas': Zum Verarbeiten und Analysieren der Daten.
14     - 'openpyxl': Zum Lesen von Excel-Dateien ('.xlsx').
15
16 ### Skript
17
18 Speichern Sie das folgende Skript in einer Datei, z.B. 'top_unterbezirke.py':
19
20     '''python
21     import pandas as pd
22
23     # Pfad zur Excel-Datei
24     excel_datei = 'Fallzahlen&HZ 2014-2023.xlsx'
25
26     # Liste der relevanten Sheet-Namen
27     jahre = list(range(2014, 2024)) # 2014 bis 2023
28     sheet_namen = [f'Fallzahlen_{jahr}' for jahr in jahre]
29
30     # Initialisiere ein leeres DataFrame zur Aggregation
31     aggregate_df = pd.DataFrame()
32
33     # Iteriere über jedes Sheet und aggregiere die Daten
34     for sheet in sheet_namen:
35         try:
36             # Lese das aktuelle Sheet

```

```

37         df = pd.read_excel(excel_datei, sheet_name=sheet)
38
39         # Überprüfe, ob die notwendigen Spalten vorhanden sind
40         if 'Bezeichnung (Bezirksregion)' in df.columns and 'Straftaten -
insgesamt-' in df.columns:
41             # Wähle die relevanten Spalten aus
42             temp_df = df[['Bezeichnung (Bezirksregion)', 'Straftaten -
insgesamt-']].copy()
43
44             # Gruppieren nach Bezirksregion und summiere die Straftaten
45             temp_agg = temp_df.groupby('Bezeichnung (Bezirksregion)',
as_index=False)['Straftaten -insgesamt-'].sum()
46
47             # Füge die aggregierten Daten zum Gesamt-DataFrame hinzu
48             aggregate_df = pd.concat([aggregate_df, temp_agg], ignore_index=
True)
49         else:
50             print(f"Warnung: Die erforderlichen Spalten sind im Sheet '{sheet
}' nicht vorhanden.")
51         except Exception as e:
52             print(f"Fehler beim Verarbeiten des Sheets '{sheet}': {e}")
53
54 # Überprüfe, ob Daten aggregiert wurden
55 if aggregate_df.empty:
56     print("Keine Daten zum Aggregieren gefunden.")
57 else:
58     # Gruppieren erneut nach Bezirksregion, um die Gesamtstraftaten über alle
Jahre zu berechnen
59     gesamt_agg = aggregate_df.groupby('Bezeichnung (Bezirksregion)', as_index
=False)['Straftaten -insgesamt-'].sum()
60
61     # Sortiere die Bezirke nach den meisten Straftaten
62     top_10 = gesamt_agg.sort_values(by='Straftaten -insgesamt-', ascending=
False).head(10)
63
64     # Ausgabe der Top 10 Unterbezirke
65     print("Top 10 Unterbezirke mit den meisten Straftaten insgesamt
(2014-2023):")
66     print(top_10.to_string(index=False))
67
68     # Optional: Speichern der Ergebnisse in einer CSV-Datei
69     top_10.to_csv('Top_10_Unterbezirke_Straftaten_2014-2023.csv', index=False
)
70     print("\nDie Ergebnisse wurden in 'Top_10_Unterbezirke_Straftaten_2014
-2023.csv' gespeichert.")
71
72
73 ### Erklärung des Skripts
74
75 1. **Importieren der Bibliotheken**:
76     - 'pandas' wird für die Datenverarbeitung verwendet.
77

```

```

78 2. **Definieren des Pfads zur Excel-Datei**:
79   - Stellen Sie sicher, dass sich die Datei 'Fallzahlen&HZ 2014-2023.xlsx'
    im selben Verzeichnis wie das Skript befindet oder geben Sie den vollständigen Pfad an.
80
81 3. **Erstellen der Liste der Sheet-Namen**:
82   - Es werden die Sheets von 'Fallzahlen_2014' bis 'Fallzahlen_2023'
    erstellt.
83
84 4. **Daten aggregieren**:
85   - Für jedes Sheet wird überprüft, ob die erforderlichen Spalten vorhanden
    sind.
86   - Die Straftaten werden pro Bezirksregion summiert.
87   - Die aggregierten Daten werden in einem Gesamt-DataFrame gesammelt.
88
89 5. **Berechnung der Gesamtstraftaten über alle Jahre**:
90   - Nach dem Aggregieren aller Sheets werden die Straftaten erneut nach
    Bezirksregion gruppiert und summiert, um die Gesamtzahl der Straftaten pro
    Bezirksregion über alle Jahre hinweg zu erhalten.
91
92 6. **Bestimmung der Top 10 Unterbezirke**:
93   - Die Bezirke werden nach der Gesamtzahl der Straftaten sortiert.
94   - Die Top 10 werden ausgewählt und angezeigt.
95
96 7. **Optional: Speichern der Ergebnisse**:
97   - Die Top 10 Liste wird zusätzlich in einer CSV-Datei namens '
    Top_10_Unterbezirke_Straftaten_2014-2023.csv' gespeichert.
98
99 ### Ausführen des Skripts
100
101 1. Stellen Sie sicher, dass das Skript und die Excel-Datei im selben
    Verzeichnis liegen oder passen Sie den Pfad zur Excel-Datei entsprechend
    an.
102
103 2. Öffnen Sie ein Terminal oder eine Eingabeaufforderung und navigieren Sie
    zu dem Verzeichnis, in dem sich das Skript befindet.
104
105 3. Führen Sie das Skript mit folgendem Befehl aus:
106
107     ''bash
108     python top_unterbezirke.py
109     ''
110
111 4. Nach der Ausführung sollten die Top 10 Unterbezirke mit den meisten
    Straftaten in der Konsole angezeigt werden und zusätzlich in der CSV-Datei
    gespeichert sein.
112
113 ### Beispielausgabe
114
115     ''
116 Top 10 Unterbezirke mit den meisten Straftaten insgesamt (2014-2023):
117 Bezeichnung (Bezirksregion)  Straftaten -insgesamt-

```

```

118         Berlin (PKS gesamt)                536695
119             Mitte                        83611
120             Alexanderplatz                19673
121             Tiergarten Süd                4491
122             Regierungsviertel            6550
123             Brunnenstraße Süd            3724
124             MV Nord                      2163
125             MV Süd                      830
126 Bezirk (Rd), nicht zuzuordnen           741
127 Stadtgebiet Berlin, nicht zuzuordnen    56425
128
129 Die Ergebnisse wurden in 'Top_10_Unterbezirke_Straftaten_2014-2023.csv'
    gespeichert.
130 '''
131
132 *(Hinweis: Die obige Ausgabe basiert auf den Beispieldaten und dient nur zur
    Veranschaulichung.)*
133
134 ### Anpassungen
135
136 - **Spaltennamen überprüfen**: Stellen Sie sicher, dass die Spaltennamen in
    der Excel-Datei exakt mit den im Skript verwendeten Namen übereinstimmen.
    Falls nicht, passen Sie die Namen im Skript entsprechend an.
137
138 - **Fehlerbehandlung erweitern**: Das Skript enthält grundlegende
    Fehlerbehandlungen. Bei Bedarf können Sie diese erweitern, um
    spezifischere Fehlerfälle abzudecken.
139
140 Falls Sie weitere Fragen haben oder Unterstützung benötigen, stehe ich gerne
    zur Verfügung!

```

Listing 182: API Response Testfall 6 Prompt 1 Ausführung 5

```

1 Natürlich, hier ist ein Python-Skript, das die beschriebenen Anforderungen
    erfüllt. Das Skript verwendet die 'pandas' Bibliothek, um die Excel-Datei
    zu verarbeiten und die gewünschten Daten zu extrahieren und zu aggregieren
    .
2
3 Stelle sicher, dass du die 'pandas' und 'openpyxl' 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&HZ 2014-2023.xlsx'
16

```

```

17 # Liste der Oberbezirke
18 oberbezirke = [
19     'Mitte',
20     'Friedrichshain-Kreuzberg',
21     'Pankow',
22     'Charlottenburg-Wilmersdorf',
23     'Spandau',
24     'Steglitz-Zehlendorf',
25     'Tempelhof-Schöneberg',
26     'Neukölln',
27     'Treptow-Köpenick',
28     'Marzahn-Hellersdorf',
29     'Lichtenberg',
30     'Reinickendorf'
31 ]
32
33 # Liste der Sheet-Namen von Fallzahlen_2014 bis Fallzahlen_2023
34 jahre = list(range(2014, 2024))
35 sheet_namen = [f'Fallzahlen_{jahr}' for jahr in jahre]
36
37 # Leere Liste zur Speicherung der DataFrames
38 df_list = []
39
40 # Iteriere über jedes Sheet und lese die Daten
41 for sheet in sheet_namen:
42     try:
43         df = pd.read_excel(excel_datei, sheet_name=sheet)
44         df_list.append(df)
45     except Exception as e:
46         print(f"Fehler beim Lesen des Sheets {sheet}: {e}")
47
48 # Zusammenführen aller DataFrames
49 gesamt_df = pd.concat(df_list, ignore_index=True)
50
51 # Entfernen der Oberbezirke und nicht zugeordneten Zeilen
52 # Bedingungen:
53 # 1. 'Bezeichnung (Bezirksregion)' nicht in oberbezirke
54 # 2. 'Bezeichnung (Bezirksregion)' enthält nicht 'nicht zuzuordnen'
55
56 # Erste Bedingung
57 bedingung_1 = ~gesamt_df['Bezeichnung (Bezirksregion)'].isin(oberbezirke)
58
59 # Zweite Bedingung: Ausschließen von Zeilen, die 'nicht zuzuordnen' enthalten
60 bedingung_2 = ~gesamt_df['Bezeichnung (Bezirksregion)'].str.contains('nicht
    zuzuordnen', case=False, na=False)
61
62 # Anwenden der Bedingungen
63 unterbezirke_df = gesamt_df[bedingung_1 & bedingung_2]
64
65 # Sicherstellen, dass die Spalte 'Straftaten -insgesamt-' numerisch ist
66 # Eventuell müssen Nicht-Numerische Werte (z.B. NaN) behandelt werden
67 unterbezirke_df['Straftaten -insgesamt-'] = pd.to_numeric(unterbezirke_df['

```

```

        Straftaten -insgesamt-'], errors='coerce').fillna(0)
68
69 # Aggregieren der Straftaten über alle Jahre hinweg
70 aggregiert_df = unterbezirke_df.groupby('Bezeichnung (Bezirksregion)')['
        Straftaten -insgesamt-'].sum().reset_index()
71
72 # Sortieren nach absteigender Anzahl der Straftaten und Auswahl der Top 10
73 top10_df = aggregiert_df.sort_values(by='Straftaten -insgesamt-', ascending=
        False).head(10)
74
75 # Optional: Zurücksetzen des Indexes
76 top10_df = top10_df.reset_index(drop=True)
77
78 # Ausgabe des finalen DataFrames
79 print(top10_df)
80
81 # Optional: Speichern des Ergebnisses in eine neue Excel-Datei
82 # top10_df.to_excel('Top10_Unterbezirke_Straftaten.xlsx', index=False)
83 '''
84
85 ### Erklärung des Skripts
86
87 1. **Importieren der benötigten Bibliotheken:**
88     '''python
89     import pandas as pd
90     '''
91
92 2. **Definieren des Pfads zur Excel-Datei und der Liste der Oberbezirke:**
93     '''python
94     excel_datei = 'Fallzahlen&HZ 2014-2023.xlsx'
95
96     oberbezirke = [
97         'Mitte',
98         'Friedrichshain-Kreuzberg',
99         'Pankow',
100        'Charlottenburg-Wilmersdorf',
101        'Spandau',
102        'Steglitz-Zehlendorf',
103        'Tempelhof-Schöneberg',
104        'Neukölln',
105        'Treptow-Köpenick',
106        'Marzahn-Hellersdorf',
107        'Lichtenberg',
108        'Reinickendorf'
109    ]
110     '''
111
112 3. **Erstellen der Liste der Sheet-Namen für die Jahre 2014 bis 2023:**
113     '''python
114     jahre = list(range(2014, 2024))
115     sheet_namen = [f'Fallzahlen_{jahr}' for jahr in jahre]
116     '''

```



```

117
118 4. **Lesen der Daten aus jedem Sheet und Zusammenführen zu einem einzigen
    DataFrame:**
119     '''python
120     df_list = []
121
122     for sheet in sheet_namen:
123         try:
124             df = pd.read_excel(excel_datei, sheet_name=sheet)
125             df_list.append(df)
126         except Exception as e:
127             print(f"Fehler beim Lesen des Sheets {sheet}: {e}")
128
129     gesamt_df = pd.concat(df_list, ignore_index=True)
130     '''
131
132 5. **Filtern der Unterbezirke:**
133     - Ausschluss der Oberbezirke.
134     - Ausschluss der Zeilen, die nicht zugeordnet sind (enthält 'nicht
      zuzuordnen').
135     '''python
136     bedingung_1 = ~gesamt_df['Bezeichnung (Bezirksregion)'].isin(oberbezirke)
137     bedingung_2 = ~gesamt_df['Bezeichnung (Bezirksregion)'].str.contains('
      nicht zuzuordnen', case=False, na=False)
138     unterbezirke_df = gesamt_df[bedingung_1 & bedingung_2]
139     '''
140
141 6. **Sicherstellen, dass die Spalte 'Straftaten -insgesamt-' numerisch ist
    und fehlende Werte behandeln:**
142     '''python
143     unterbezirke_df['Straftaten -insgesamt-'] = pd.to_numeric(unterbezirke_df[
      'Straftaten -insgesamt-'], errors='coerce').fillna(0)
144     '''
145
146 7. **Aggregieren der Straftaten über alle Jahre und Ermitteln der Top 10
    Unterbezirke:**
147     '''python
148     aggregiert_df = unterbezirke_df.groupby('Bezeichnung (Bezirksregion)')['
      Straftaten -insgesamt-'].sum().reset_index()
149     top10_df = aggregiert_df.sort_values(by='Straftaten -insgesamt-',
      ascending=False).head(10)
150     top10_df = top10_df.reset_index(drop=True)
151     '''
152
153 8. **Ausgabe des finalen DataFrames:**
154     '''python
155     print(top10_df)
156     '''
157
158 9. **Optional: Speichern des Ergebnisses in eine neue Excel-Datei:**
159     '''python
160     # top10_df.to_excel('Top10_Unterbezirke_Straftaten.xlsx', index=False)

```

```

161     '''
162
163     ### Hinweise
164
165     - **Fehlerbehandlung:** Das Skript enthält einen 'try-except' Block, um
      eventuelle Fehler beim Lesen der einzelnen Sheets abzufangen und
      auszugeben, ohne das gesamte Skript zu stoppen.
166
167     - **Spaltennamen prüfen:** Stelle sicher, dass die Spaltennamen in deinen
      Excel-Sheets exakt mit den im Skript verwendeten Namen übereinstimmen.
      Falls die Spalte beispielsweise 'Straftaten \n-insgesamt-' anstelle von '
      Straftaten -insgesamt-' heißt, musst du den Spaltennamen entsprechend
      anpassen.
168
169     - **Leistung:** Bei sehr großen Excel-Dateien kann das Laden und Verarbeiten
      einige Zeit in Anspruch nehmen. Stelle sicher, dass dein System
      ausreichend Ressourcen hat.
170
171     - **Datentypen:** Die Konvertierung der 'Straftaten -insgesamt-' Spalte zu
      numerischen Werten ist wichtig, um korrekte Aggregationen durchführen zu k
      önnen. Nicht-numerische Werte werden hierbei als 0 behandelt.
172
173     Dieses Skript sollte dir dabei helfen, die gewünschten Unterbezirke zu
      identifizieren und die Top 10 mit den meisten Straftaten zu ermitteln.
      Passe das Skript bei Bedarf an deine spezifischen Datenstrukturen an.

```

Listing 183: API Response Testfall 6 Prompt 2 Ausführung 1

```

1 Hier ist ein Python-Skript, das die beschriebenen Anforderungen erfüllt. Das
  Skript verwendet die Bibliothek 'pandas', um die Excel-Datei zu lesen, die
  Daten zu verarbeiten und die gewünschten Ergebnisse zu erzielen.
2
3 '''python
4 import pandas as pd
5
6 # Pfad zur Excel-Datei
7 excel_file = 'Fallzahlen&HZ 2014-2023.xlsx'
8
9 # Liste der Oberbezirke
10 oberbezirke = [
11     'Mitte',
12     'Friedrichshain-Kreuzberg',
13     'Pankow',
14     'Charlottenburg-Wilmersdorf',
15     'Spandau',
16     'Steglitz-Zehlendorf',
17     'Tempelhof-Schöneberg',
18     'Neukölln',
19     'Treptow-Köpenick',
20     'Marzahn-Hellersdorf',
21     'Lichtenberg',
22     'Reinickendorf'
23 ]

```

```

24
25 # Generiere die Liste der Sheet-Namen Fallzahlen_2014 bis Fallzahlen_2023
26 jahre = range(2014, 2024)
27 sheet_names = [f'Fallzahlen_{jahr}' for jahr in jahre]
28
29 # Liste zur Speicherung der einzelnen DataFrames
30 df_list = []
31
32 # Iteriere über alle Sheet-Namen und lese die Daten
33 for sheet in sheet_names:
34     try:
35         df = pd.read_excel(excel_file, sheet_name=sheet)
36         df_list.append(df)
37         print(f'Sheet {sheet} erfolgreich gelesen.')
38     except Exception as e:
39         print(f'Fehler beim Lesen des Sheets {sheet}: {e}')
40
41 # Zusammenführen aller DataFrames in einen einzigen DataFrame
42 combined_df = pd.concat(df_list, ignore_index=True)
43 print('Alle Sheets wurden erfolgreich zusammengeführt.')
44
45 # Anzeige der Spalten zur Überprüfung (optional)
46 # print(combined_df.columns)
47
48 # Filtern der Unterbezirke:
49 # - Entfernen der Oberbezirke
50 # - Entfernen von Zeilen, die "nicht zuzuordnen" enthalten
51
52 # Bedingung für Unterbezirke:
53 # - 'Bezeichnung (Bezirksregion)' ist nicht in der Liste der Oberbezirke
54 # - 'Bezeichnung (Bezirksregion)' enthält nicht 'nicht zuzuordnen' (case
    insensitive)
55
56 filtered_df = combined_df[
57     (~combined_df['Bezeichnung (Bezirksregion)'].isin(oberbezirke)) &
58     (~combined_df['Bezeichnung (Bezirksregion)'].str.contains('nicht
    zuzuordnen', case=False, na=False))
59 ]
60
61 print(f'Anzahl der Zeilen nach Filtern: {filtered_df.shape[0]}')
62
63 # Sicherstellen, dass die Spalte 'Straftaten -insgesamt-' numerisch ist
64 # Entfernen von Tausenderpunkten und Umwandeln in Integer
65 filtered_df['Straftaten -insgesamt-'] = filtered_df['Straftaten -insgesamt-']
    .astype(str).str.replace('.', '').astype(int)
66
67 # Aggregieren der Straftaten pro Unterbezirk über alle Jahre hinweg
68 aggregated_df = filtered_df.groupby('Bezeichnung (Bezirksregion)', as_index=
    False)['Straftaten -insgesamt-'].sum()
69
70 # Sortieren nach der aggregierten Anzahl der Straftaten in absteigender
    Reihenfolge

```

```

71 sorted_df = aggregated_df.sort_values(by='Straftaten -insgesamt-', ascending=
    False)
72
73 # Auswahl der Top 10 Unterbezirke mit den meisten Straftaten
74 top_10_df = sorted_df.head(10)
75
76 # Auswahl der gewünschten Spalten
77 final_df = top_10_df[['Bezeichnung (Bezirksregion)', 'Straftaten -insgesamt-']
    []]
78
79 # Optional: Umbenennen der Spalten für bessere Verständlichkeit
80 final_df.rename(columns={
81     'Bezeichnung (Bezirksregion)': 'Unterbezirk',
82     'Straftaten -insgesamt-': 'Gesamtstraftaten'
83 }, inplace=True)
84
85 # Anzeige des finalen DataFrames
86 print('Top 10 Unterbezirke mit den meisten Straftaten:')
87 print(final_df)
88
89 # Optional: Speichern des finalen DataFrames in eine neue Excel-Datei
90 # final_df.to_excel('Top_10_Unterbezirke_Straftaten.xlsx', index=False)
91 '''
92
93 ### Erklärung des Skripts
94
95 1. **Importieren der benötigten Bibliothek:**
96 - 'pandas' wird verwendet, um Excel-Dateien zu lesen und Daten zu
    verarbeiten.
97
98 2. **Definieren des Pfads zur Excel-Datei:**
99 - Stellen Sie sicher, dass sich die Datei 'Fallzahlen&HZ 2014-2023.xlsx'
    im gleichen Verzeichnis wie das Skript befindet oder passen Sie den Pfad
    entsprechend an.
100
101 3. **Erstellen einer Liste der Oberbezirke:**
102 - Diese Liste wird verwendet, um die Oberbezirke aus den Daten zu filtern.
103
104 4. **Generieren der Sheet-Namen:**
105 - Die Sheets sind von 'Fallzahlen_2014' bis 'Fallzahlen_2023'. Diese
    werden dynamisch anhand der Jahreszahlen generiert.
106
107 5. **Lesen und Zusammenführen der Daten:**
108 - Das Skript iteriert über alle Sheet-Namen, liest die Daten und fügt sie
    zu einer Liste von DataFrames zusammen.
109 - Anschließend werden alle DataFrames mit 'pd.concat' zu einem einzigen
    DataFrame kombiniert.
110
111 6. **Filtern der Daten:**
112 - Entfernen der Zeilen, die zu den Oberbezirken gehören.
113 - Entfernen der Zeilen, die "nicht zuzuordnen" enthalten, um
    sicherzustellen, dass nur gültige Unterbezirke berücksichtigt werden.

```

```

114
115 7. **Datenbereinigung:**
116     - Sicherstellen, dass die Spalte 'Straftaten -insgesamt-' numerisch ist.
      Falls die Zahlen Tausenderpunkte enthalten, werden diese entfernt und die
      Spalte in Ganzzahlen umgewandelt.
117
118 8. **Aggregation und Sortierung:**
119     - Aggregieren der Gesamtanzahl der Straftaten pro Unterbezirk über alle
      Jahre hinweg.
120     - Sortieren der Ergebnisse in absteigender Reihenfolge, um die
      Unterbezirke mit den meisten Straftaten zu identifizieren.
121
122 9. **Auswahl der Top 10:**
123     - Auswahl der obersten 10 Unterbezirke basierend auf der aggregierten
      Anzahl der Straftaten.
124
125 10. **Bereinigung des finalen DataFrames:**
126     - Auswahl der gewünschten Spalten und Umbenennung für bessere Verstä
      ndlichkeit.
127
128 11. **Ausgabe:**
129     - Das Skript gibt den finalen DataFrame mit den Top 10 Unterbezirken und
      deren Gesamtanzahl der Straftaten aus.
130     - Optional kann das Ergebnis auch in eine neue Excel-Datei gespeichert
      werden, indem die entsprechende Zeile entkommentiert wird.
131
132 ### Hinweise
133
134 - **Fehlerbehandlung beim Lesen der Sheets:**
135     - Das Skript enthält eine 'try-except'-Struktur, um Fehler beim Lesen
      einzelner Sheets zu handhaben. So wird das Skript nicht abgebrochen, wenn
      ein bestimmtes Sheet fehlt oder einen Fehler enthält.
136
137 - **Anpassung der Spaltennamen:**
138     - Stellen Sie sicher, dass die Spaltennamen in Ihren Excel-Sheets exakt mit
      denen im Skript übereinstimmen. Falls nötig, passen Sie die Namen im
      Skript an.
139
140 - **Performance:**
141     - Bei sehr großen Excel-Dateien kann das Lesen und Verarbeiten einige Zeit
      in Anspruch nehmen. Optimierungen können je nach Bedarf vorgenommen werden
      .
142
143 - **Abhängigkeiten:**
144     - Stellen Sie sicher, dass 'pandas' installiert ist. Falls nicht, können
      Sie es mit 'pip install pandas' installieren.
145
146 Dieses Skript bietet eine umfassende Lösung für die beschriebenen
      Anforderungen und kann nach Bedarf weiter angepasst werden.

```

Listing 184: API Response Testfall 6 Prompt 2 Ausführung 2

```

1 Natürlich, ich helfe Ihnen gerne dabei, ein Python-Skript zu erstellen, das

```

Ihre Excel-Daten verarbeitet und die gewünschten Analysen durchführt.
Unten finden Sie ein ausführliches Skript mit Kommentaren, die die
einzelnen Schritte erläutern.

Voraussetzungen

Stellen Sie sicher, dass die folgenden Python-Bibliotheken installiert sind:

- 'pandas' zum Datenmanagement
- 'openpyxl' als Engine zum Lesen von Excel-Dateien

Sie können diese Bibliotheken mit 'pip' installieren, falls Sie sie noch
nicht installiert haben:

```
'''bash
pip install pandas openpyxl
'''
```

Python-Skript

```
'''python
import pandas as pd

# Pfad zur Excel-Datei
excel_datei = 'Fallzahlen&HZ 2014-2023.xlsx'

# Liste der Oberbezirke, die ausgeschlossen werden sollen
oberbezirke = [
    'Mitte', 'Friedrichshain-Kreuzberg', 'Pankow',
    'Charlottenburg-Wilmersdorf', 'Spandau', 'Steglitz-Zehlendorf',
    'Tempelhof-Schöneberg', 'Neukölln', 'Treptow-Köpenick',
    'Marzahn-Hellersdorf', 'Lichtenberg', 'Reinickendorf'
]

# Erstellen einer Liste der Sheet-Namen von 2014 bis 2023
jahreszahlen = range(2014, 2024) # 2014 bis 2023 inklusive
sheet_namen = [f'Fallzahlen_{jahr}' for jahr in jahreszahlen]

# Lesen aller relevanten Sheets und Speichern in einer Liste von DataFrames
dataframes = []
for sheet in sheet_namen:
    try:
        df = pd.read_excel(excel_datei, sheet_name=sheet, engine='openpyxl')
        dataframes.append(df)
        print(f"Sheet '{sheet}' erfolgreich gelesen.")
    except Exception as e:
        print(f"Fehler beim Lesen von Sheet '{sheet}': {e}")

# Zusammenführen aller DataFrames zu einem einzigen DataFrame
if dataframes:
    kombiniertes_df = pd.concat(dataframes, ignore_index=True)
    print("Alle Sheets erfolgreich zusammengeführt.")
```

```

50 else:
51     raise ValueError("Keine Daten zum Verarbeiten gefunden.")
52
53 # Überprüfen der Spaltennamen
54 print("Spaltennamen im kombinierten DataFrame:")
55 print(kombiniertes_df.columns.tolist())
56
57 # Bereinigung der Spaltennamen (Entfernen von Zeilenumbrüchen und Leerzeichen
58     )
59 kombiniertes_df.columns = kombiniertes_df.columns.str.replace('\n', ' ',
60     regex=True).str.strip()
61
62 # Überprüfen der bereinigten Spaltennamen
63 print("Bereinigte Spaltennamen:")
64 print(kombiniertes_df.columns.tolist())
65
66 # Filtern der Unterbezirke (Ausschließen der Oberbezirke und nicht
67     zugeordneten Einträge)
68 unterbezirke_df = kombiniertes_df[~kombiniertes_df['Bezeichnung (
69     Bezirksregion)'].isin(oberbezirke)]
70 print(f"Anzahl der Unterbezirke nach Filter: {unterbezirke_df.shape[0]}")
71
72 # Weitere Filterung: Entfernen von nicht zugeordneten Zeilen
73 # Angenommen, nicht zugeordnete Zeilen enthalten bestimmte Schlüsselwörter
74 nicht_zugeordnet_keywords = ['nicht zuzuordnen', 'Berlin (PKS gesamt)']
75 unterbezirke_df = unterbezirke_df[~unterbezirke_df['Bezeichnung (
76     Bezirksregion)'].str.contains('|'.join(nicht_zugeordnet_keywords), case=
77     False, na=False)]
78 print(f"Anzahl der Unterbezirke nach Entfernen nicht zugeordneter Zeilen: {
79     unterbezirke_df.shape[0]}")
80
81 # Sicherstellen, dass die Spalte 'Straftaten -insgesamt-' vorhanden ist
82 straftaten_spalte = 'Straftaten -insgesamt-'
83 if straftaten_spalte not in unterbezirke_df.columns:
84     raise ValueError(f"Die Spalte '{straftaten_spalte}' wurde nicht gefunden.
85     Bitte überprüfen Sie die Spaltennamen.")
86
87 # Gruppieren nach Unterbezirk und Summieren der Straftaten
88 aggregiertes_df = unterbezirke_df.groupby('Bezeichnung (Bezirksregion)')[
89     straftaten_spalte].sum().reset_index()
90
91 # Sortieren nach der Anzahl der Straftaten in absteigender Reihenfolge
92 aggregiertes_df = aggregiertes_df.sort_values(by=straftaten_spalte, ascending
93     =False)
94
95 # Auswahl der Top 10 Unterbezirke mit den meisten Straftaten
96 top10_df = aggregiertes_df.head(10)
97
98 # Finale Darstellung mit nur den gewünschten Spalten
99 final_df = top10_df[['Bezeichnung (Bezirksregion)', straftaten_spalte]]
100
101 # Ausgabe des finalen DataFrames

```

```

92 print("Top 10 Unterbezirke mit den meisten Straftaten:")
93 print(final_df)
94
95 # Optional: Speichern des finalen DataFrames in eine neue Excel-Datei
96 final_df.to_excel('Top10_Unterbezirke_Straftaten.xlsx', index=False)
97 print("Das finale DataFrame wurde in 'Top10_Unterbezirke_Straftaten.xlsx'
    gespeichert.")
98 '''
99
100 ### Erklärung der Schritte
101
102 1. **Import der Bibliotheken:**
103     - 'pandas' wird für die Datenmanipulation verwendet.
104
105 2. **Definieren des Excel-Dateipfades und der Oberbezirke:**
106     - Legen Sie den Pfad zur Excel-Datei fest.
107     - Listen Sie alle Oberbezirke auf, die ausgeschlossen werden sollen.
108
109 3. **Generierung der Sheet-Namen:**
110     - Erstellen Sie eine Liste der Sheet-Namen von 'Fallzahlen_2014' bis '
    Fallzahlen_2023'.
111
112 4. **Lesen der Excel-Sheets:**
113     - Iterieren Sie über alle Sheet-Namen und lesen Sie jeden Sheet in ein
    DataFrame ein.
114     - Bei Fehlern wird eine Meldung ausgegeben, aber das Skript versucht, mit
    den verfügbaren Daten fortzufahren.
115
116 5. **Zusammenführen der DataFrames:**
117     - Kombinieren Sie alle gelesenen DataFrames zu einem einzigen DataFrame.
118
119 6. **Bereinigung der Spaltennamen:**
120     - Entfernen Sie Zeilenumbrüche und überflüssige Leerzeichen aus den
    Spaltennamen, um Konsistenz zu gewährleisten.
121
122 7. **Filtern der Unterbezirke:**
123     - Entfernen Sie alle Zeilen, die zu den definierten Oberbezirken gehören.
124     - Zusätzlich werden Zeilen entfernt, die nicht zugeordnet sind, basierend
    auf spezifischen Schlüsselwörtern.
125
126 8. **Aggregation der Straftaten:**
127     - Gruppieren Sie die Daten nach Unterbezirken und summieren Sie die
    Gesamtzahl der Straftaten über alle Jahre hinweg.
128
129 9. **Ermittlung der Top 10 Unterbezirke:**
130     - Sortieren Sie die Unterbezirke nach der Anzahl der Straftaten in
    absteigender Reihenfolge und wählen Sie die Top 10 aus.
131
132 10. **Finalisierung des DataFrames:**
133     - Wählen Sie nur die erforderlichen Spalten aus und speichern Sie die
    Ergebnisse.
134     - Optional können Sie das finale DataFrame in eine neue Excel-Datei

```



```

135     exportieren.
136 ### Hinweise
137
138 - **Spaltennamen überprüfen:** Stellen Sie sicher, dass die Spaltennamen in
    Ihrem Excel genau mit den im Skript verwendeten Namen übereinstimmen.
    Eventuell müssen Sie die Namen anpassen, insbesondere wenn Zeilenumbrüche
    oder unerwartete Leerzeichen vorhanden sind.
139
140 - **Fehlerbehandlung:** Das Skript enthält grundlegende Fehlerbehandlungen,
    wie das Überprüfen der Existenz der notwendigen Spalten und das Fortsetzen
    bei fehlenden Sheets. Je nach Datenqualität und -struktur können weitere
    Anpassungen erforderlich sein.
141
142 - **Anpassung der Schlüsselwörter:** Falls die nicht zugeordneten Zeilen
    andere Schlüsselwörter enthalten, müssen diese in der Liste '
    nicht_zugeordnet_keywords' ergänzt werden.
143
144 Ich hoffe, dieses Skript erfüllt Ihre Anforderungen. Bei weiteren Fragen oder
    Anpassungswünschen stehe ich Ihnen gerne zur Verfügung!

```

Listing 185: API Response Testfall 6 Prompt 2 Ausführung 3

```

1 Hier ist ein Python-Skript, das Ihre Anforderungen erfüllt. Das Skript
  verwendet die Bibliothek 'pandas', um die Excel-Datei zu verarbeiten, die
  Daten zu filtern und die gewünschten Ergebnisse zu aggregieren.
2
3 Stellen Sie sicher, dass Sie 'pandas' und 'openpyxl' installiert haben. Falls
  nicht, können Sie sie mit 'pip' installieren:
4
5 '''bash
6 pip install pandas openpyxl
7 '''
8
9 ### Python-Skript
10
11 '''python
12 import pandas as pd
13
14 # Dateipfad zur Excel-Datei
15 file_path = 'Fallzahlen&HZ 2014-2023.xlsx'
16
17 # Liste der Oberbezirke
18 oberbezirke = [
19     'Mitte',
20     'Friedrichshain-Kreuzberg',
21     'Pankow',
22     'Charlottenburg-Wilmersdorf',
23     'Spandau',
24     'Steglitz-Zehlendorf',
25     'Tempelhof-Schöneberg',
26     'Neukölln',
27     'Treptow-Köpenick',

```

```

28     'Marzahn-Hellersdorf',
29     'Lichtenberg',
30     'Reinickendorf'
31 ]
32
33 # Generiere die Liste der Sheet-Namen von 2014 bis 2023
34 sheet_names = [f'Fallzahlen_{year}' for year in range(2014, 2024)]
35
36 # Lese alle relevanten Sheets ein und speichere sie in einer Liste
37 dfs = []
38 for sheet in sheet_names:
39     try:
40         df = pd.read_excel(file_path, sheet_name=sheet)
41         dfs.append(df)
42     except ValueError:
43         print(f"Sheet {sheet} nicht gefunden in der Datei.")
44
45 # Kombiniere alle DataFrames zu einem einzigen DataFrame
46 combined_df = pd.concat(dfs, ignore_index=True)
47
48 # Bereinige die Spaltennamen: Entferne Zeilenumbrüche und führende/trailende
    Leerzeichen
49 combined_df.columns = combined_df.columns.str.replace('\n', ' ').str.strip()
50
51 # Identifiziere die Spalte für "Straftaten -insgesamt-"
52 # Diese Annahme basiert auf den bereitgestellten Beispieldaten
53 straftaten_spalte = 'Straftaten -insgesamt-'
54
55 if straftaten_spalte not in combined_df.columns:
56     # Falls die Spalte nicht exakt gefunden wird, suche ähnlich benannte
        Spalten
57     mögliche_spalten = [col for col in combined_df.columns if 'Straftaten' in
        col and 'insgesamt' in col.lower()]
58     if mögliche_spalten:
59         straftaten_spalte = mögliche_spalten[0]
60     else:
61         raise ValueError('Die Spalte für "Straftaten -insgesamt-" wurde nicht
        gefunden.')
```

```

62
63 # Filtere nur die Unterbezirke:
64 # - Schließe die Oberbezirke aus
65 # - Schließe Zeilen aus, die "nicht zuzuordnen" enthalten
66 unterbezirke_df = combined_df[
67     (~combined_df['Bezeichnung (Bezirksregion)'].isin(oberbezirke)) &
68     (~combined_df['Bezeichnung (Bezirksregion)'].str.contains('nicht
        zuzuordnen', case=False, na=False))
69 ]
70
71 # Stelle sicher, dass die Straftaten-Spalte numerisch ist
72 unterbezirke_df[straftaten_spalte] = pd.to_numeric(unterbezirke_df[
        straftaten_spalte], errors='coerce').fillna(0)
73

```

```

74 # Aggregiere die Straftaten über alle Jahre hinweg pro Unterbezirk
75 aggregated_df = unterbezirke_df.groupby('Bezeichnung (Bezirksregion)')[
    straftaten_spalte].sum().reset_index()
76
77 # Ermittle die Top 10 Unterbezirke mit den meisten Straftaten
78 top10_df = aggregated_df.sort_values(by=straftaten_spalte, ascending=False).
    head(10)
79
80 # Benenne die Spalten für den finalen DataFrame um (optional)
81 final_df = top10_df.rename(columns={
82     'Bezeichnung (Bezirksregion)': 'Unterbezirk',
83     straftaten_spalte: 'Straftaten_Gesamt'
84 })
85
86 # Optional: Setze die Spalten in eine bestimmte Reihenfolge
87 final_df = final_df[['Unterbezirk', 'Straftaten_Gesamt']]
88
89 # Ausgabe des finalen DataFrames
90 print(final_df)
91
92 # Optional: Speichere das Ergebnis in eine neue Excel-Datei
93 # final_df.to_excel('Top10_Unterbezirke_Straftaten.xlsx', index=False)
94 '''
95
96 ### Erklärung der einzelnen Schritte
97
98 1. **Import der Bibliotheken**: Das Skript verwendet 'pandas' zur
    Datenverarbeitung.
99
100 2. **Dateipfad und Oberbezirke definieren**: Der Pfad zur Excel-Datei und die
    Liste der Oberbezirke werden festgelegt.
101
102 3. **Sheet-Namen generieren**: Erzeugt eine Liste der Sheet-Namen von '
    Fallzahlen_2014' bis 'Fallzahlen_2023'.
103
104 4. **Einlesen der Excel-Sheets**: Iteriert über die Sheet-Namen, liest jedes
    Sheet ein und fügt es der Liste 'dfs' hinzu. Falls ein Sheet nicht
    gefunden wird, gibt das Skript eine Warnung aus.
105
106 5. **Kombinieren der DataFrames**: Alle einzelnen DataFrames werden zu einem
    großen DataFrame 'combined_df' zusammengeführt.
107
108 6. **Bereinigen der Spaltennamen**: Entfernt Zeilenumbrüche und unnötige
    Leerzeichen aus den Spaltennamen, um eine konsistente Datenverarbeitung zu
    gewährleisten.
109
110 7. **Identifizieren der Straftaten-Spalte**: Überprüft, ob die Spalte '
    Straftaten -insgesamt-' vorhanden ist. Falls nicht, sucht das Skript nach
    einer ähnlichen Spalte.
111
112 8. **Filtern der Unterbezirke**:
113     - Schließt alle Oberbezirke aus der Liste aus.

```

```

114     - Entfernt alle Zeilen, die den Begriff "nicht zuzuordnen" enthalten.
115
116 9. **Konvertieren der Straftaten-Spalte in numerische Werte**: Stellt sicher,
    dass die Straftatenzahlen als numerische Werte vorliegen, und füllt
    fehlende Werte mit 0.
117
118 10. **Aggregieren der Straftaten**: Summiert die Straftaten über alle Jahre
    hinweg für jeden Unterbezirk.
119
120 11. **Ermitteln der Top 10 Unterbezirke**: Sortiert die Unterbezirke nach der
    Gesamtzahl der Straftaten in absteigender Reihenfolge und wählt die Top
    10 aus.
121
122 12. **Formatieren des finalen DataFrames**: Benennt die Spalten um und ordnet
    sie für eine bessere Lesbarkeit.
123
124 13. **Ausgabe und optionales Speichern**: Gibt das finale Ergebnis aus und
    bietet die Möglichkeit, es in eine neue Excel-Datei zu speichern.
125
126 ### Beispielausgabe
127
128 Die Ausgabe könnte wie folgt aussehen (Beispielwerte):
129
130 '''
131         Unterbezirk  Straftaten_Gesamt
132 0           Kreuzberg           150000
133 1           Tempelhof           140000
134 2           Neukölln           130000
135 3           Lichtenberg        120000
136 4   Friedrichshain-Kreuzberg  110000
137 5           Treptow-Köpenick    100000
138 6           Steglitz-Zehlendorf  90000
139 7           Marzahn-Hellersdorf  80000
140 8           Reinickendorf       70000
141 9           Pankow             60000
142 '''
143
144 ### Hinweise
145
146 - **Spaltennamen**: Stellen Sie sicher, dass die Spaltennamen in Ihren Excel-
    Sheets mit denen im Skript übereinstimmen. Falls nicht, müssen Sie die
    Namen im Skript entsprechend anpassen.
147
148 - **Fehlende Daten**: Das Skript geht davon aus, dass alle relevanten Sheets
    vorhanden sind. Falls einige Sheets fehlen, wird eine Warnung ausgegeben,
    und das Skript fährt mit den verfügbaren Daten fort.
149
150 - **Speichern des Ergebnisses**: Die letzte Zeile im Skript, die das Ergebnis
    in eine neue Excel-Datei speichert, ist auskommentiert. Entfernen Sie das
    Kommentarzeichen ('#'), wenn Sie das Ergebnis speichern möchten.
151
152 Dieses Skript sollte Ihnen helfen, die gewünschten Daten effizient zu

```

verarbeiten und die Top 10 Unterbezirke mit den meisten Straftaten zu ermitteln.

Listing 186: API Response Testfall 6 Prompt 2 Ausführung 4

```
1 Natürlich, hier ist ein Python-Skript, das die beschriebenen Anforderungen
  erfüllt. Das Skript verwendet die 'pandas'-Bibliothek, um die Excel-Datei
  zu lesen, die Daten zu verarbeiten und die gewünschten Ergebnisse zu
  erzielen.
2
3 '''python
4 import pandas as pd
5
6 # Pfad zur Excel-Datei
7 excel_datei = 'Fallzahlen&HZ 2014-2023.xlsx'
8
9 # Liste der Oberbezirke
10 oberbezirke = [
11     'Mitte', 'Friedrichshain-Kreuzberg', 'Pankow', 'Charlottenburg-
    Wilmersdorf',
12     'Spandau', 'Steglitz-Zehlendorf', 'Tempelhof-Schöneberg', 'Neukölln',
13     'Treptow-Köpenick', 'Marzahn-Hellersdorf', 'Lichtenberg', 'Reinickendorf'
14 ]
15
16 # Liste der relevanten Sheet-Namen
17 sheet_namen = [f'Fallzahlen_{jahr}' for jahr in range(2014, 2024)]
18
19 # Alle relevanten Sheets einlesen und in eine Liste von DataFrames speichern
20 daten_list = []
21 for sheet in sheet_namen:
22     try:
23         df = pd.read_excel(excel_datei, sheet_name=sheet)
24         daten_list.append(df)
25     except Exception as e:
26         print(f"Fehler beim Lesen des Sheets {sheet}: {e}")
27
28 # Alle DataFrames zu einem einzigen DataFrame zusammenführen
29 gesamt_df = pd.concat(daten_list, ignore_index=True)
30
31 # Anzeige der Spaltennamen zur Überprüfung (optional)
32 # print(gesamt_df.columns)
33
34 # Filtern der Unterbezirke:
35 # - Ausschließen der Oberbezirke
36 # - Ausschließen von nicht zugeordneten Zeilen
37 filtered_df = gesamt_df[
38     (~gesamt_df['Bezeichnung (Bezirksregion)'].isin(oberbezirke)) &
39     (~gesamt_df['Bezeichnung (Bezirksregion)'].str.contains('nicht zuzuordnen'
40     ', case=False, na=False))
41 ]
42
43 # Sicherstellen, dass die 'Straftaten -insgesamt-' Spalte numerisch ist
44 filtered_df['Straftaten -insgesamt-'] = pd.to_numeric(filtered_df['Straftaten
```

```

    -insgesamt-'], errors='coerce')
44
45 # Entfernen von Zeilen mit NaN in 'Straftaten -insgesamt-'
46 filtered_df = filtered_df.dropna(subset=['Straftaten -insgesamt-'])
47
48 # Aggregieren der Straftaten pro Unterbezirk
49 aggregiert_df = filtered_df.groupby('Bezeichnung (Bezirksregion)')['
    Straftaten -insgesamt-'].sum().reset_index()
50
51 # Top 10 Unterbezirke mit den meisten Straftaten
52 top_10_df = aggregiert_df.sort_values(by='Straftaten -insgesamt-', ascending=
    False).head(10)
53
54 # Optional: Zur besseren Darstellung sortieren
55 top_10_df = top_10_df.sort_values(by='Straftaten -insgesamt-', ascending=
    False)
56
57 # Finale Ausgabe
58 print("Top 10 Unterbezirke mit den meisten Straftaten:")
59 print(top_10_df)
60
61 # Optional: Speichern des Ergebnisses in eine neue Excel-Datei
62 # top_10_df.to_excel('Top_10_Unterbezirke_Straftaten.xlsx', index=False)
63 '''
64
65 ### Erklärung der Schritte:
66
67 1. **Importieren der notwendigen Bibliothek:**
68     - 'pandas' wird verwendet, um die Excel-Datei zu lesen und die Daten zu
        verarbeiten.
69
70 2. **Festlegen des Pfads zur Excel-Datei und der Liste der Oberbezirke:**
71     - 'excel_datei' enthält den Namen der Excel-Datei.
72     - 'oberbezirke' ist eine Liste der Oberbezirke, die ausgeschlossen werden
        sollen.
73
74 3. **Generieren der Liste der relevanten Sheet-Namen:**
75     - Die Sheets werden von 'Fallzahlen_2014' bis 'Fallzahlen_2023' erwartet.
76
77 4. **Einlesen aller relevanten Sheets:**
78     - Mit einer Schleife werden alle Sheets nacheinander eingelesen und in der
        Liste 'daten_list' gespeichert.
79     - Fehler beim Lesen einzelner Sheets werden ausgegeben, ohne das gesamte
        Skript zu unterbrechen.
80
81 5. **Zusammenführen aller DataFrames:**
82     - Alle DataFrames aus 'daten_list' werden zu einem einzigen DataFrame '
        gesamt_df' zusammengeführt.
83
84 6. **Filtern der Unterbezirke:**
85     - Ausschließen der Oberbezirke mittels 'isin'.
86     - Ausschließen von Zeilen, die den Begriff nicht zuzuordnen enthalten (

```

```

unabhängig von Groß-/Kleinschreibung).
87
88 7. **Sicherstellen der numerischen Werte in der Straftaten-Spalte:**
89 - Die Spalte 'Straftaten -insgesamt-' wird in numerische Werte
    umgewandelt.
90 - Zeilen mit ungültigen Werten werden entfernt.
91
92 8. **Aggregieren der Straftaten pro Unterbezirk:**
93 - Gruppieren nach 'Bezeichnung (Bezirksregion)' und Summieren der
    Straftaten.
94
95 9. **Ermitteln der Top 10 Unterbezirke:**
96 - Sortieren nach der aggregierten Summe der Straftaten in absteigender
    Reihenfolge.
97 - Auswahl der ersten 10 Einträge.
98
99 10. **Ausgabe des finalen DataFrames:**
100 - Die Top 10 Unterbezirke werden im Terminal angezeigt.
101 - Optional kann das Ergebnis in eine neue Excel-Datei gespeichert werden,
    indem die auskommentierte Zeile aktiviert wird.
102
103 ### Hinweise:
104
105 - Stelle sicher, dass die Excel-Datei sich im selben Verzeichnis wie das
    Skript befindet oder gib den vollständigen Pfad zur Datei an.
106 - Falls die Spaltennamen leicht abweichen, überprüfe sie mit 'print(gesamt_df
    .columns)' und passe sie gegebenenfalls im Skript an.
107 - Um das Skript auszuführen, benötigst du die 'pandas'-Bibliothek.
    Installiere sie ggf. mit 'pip install pandas' und auch 'openpyxl' für das
    Lesen von Excel-Dateien: 'pip install openpyxl'.
108 - Bei sehr großen Excel-Dateien könnte das Einlesen und Verarbeiten etwas
    Zeit in Anspruch nehmen.
109
110 Ich hoffe, dieses Skript hilft dir weiter! Wenn du weitere Anpassungen benö-
    tigtst oder Fragen hast, lass es mich wissen.

```

Listing 187: API Response Testfall 6 Prompt 2 Ausführung 5

```

1  '''python
2  import pandas as pd
3
4  # 1. Definiere den Dateinamen
5  excel_file = 'Fallzahlen&HZ 2014-2023.xlsx'
6
7  # 2. Liste der Oberbezirke
8  oberbezirke = [
9      'Mitte', 'Friedrichshain-Kreuzberg', 'Pankow', 'Charlottenburg-
    Wilmersdorf',
10     'Spandau', 'Steglitz-Zehlendorf', 'Tempelhof-Schöneberg', 'Neukölln',
11     'Treptow-Köpenick', 'Marzahn-Hellersdorf', 'Lichtenberg', 'Reinickendorf'
12 ]
13
14 # 3. Lese alle Sheets ein und speichere sie in einer Liste von DataFrames

```

```

15 # Annahme: Die Sheets heißen 'Fallzahlen_2014' bis 'Fallzahlen_2023'
16 jahre = range(2014, 2024)
17 dataframes = []
18
19 for jahr in jahre:
20     sheet_name = f'Fallzahlen_{jahr}'
21     try:
22         df = pd.read_excel(excel_file, sheet_name=sheet_name)
23         df['Jahr'] = jahr # Füge eine Spalte für das Jahr hinzu
24         dataframes.append(df)
25     except Exception as e:
26         print(f"Fehler beim Lesen des Sheets '{sheet_name}': {e}")
27
28 # 4. Füge alle DataFrames zu einem einzigen DataFrame zusammen
29 gesamt_df = pd.concat(dataframes, ignore_index=True)
30
31 # 5. Entferne Zeilen, die Oberbezirke, 'Berlin (PKS gesamt)' oder nicht
    zuzuordnende Einträge enthalten
32 # Definiere die Namen der nicht zuzuordnenden Einträge
33 nicht_zugeordnet = [
34     'Bezirk (Rd), nicht zuzuordnen',
35     'Stadtgebiet Berlin, nicht zuzuordnen',
36     'Berlin (PKS gesamt)'
37 ]
38
39 # Filtere die DataFrame, um nur Unterbezirke zu behalten
40 unterbezirk_df = gesamt_df[~gesamt_df['Bezeichnung (Bezirksregion)'].isin(
    oberbezirke + nicht_zugeordnet)]
41
42 # 6. Summiere die Straftaten über alle Jahre hinweg für jeden Unterbezirk
43 # Achte auf den genauen Spaltennamen, möglicherweise enthält er einen
    Zeilenumbruch
44 # Daher benutze hier eine Rohzeichenkette oder ersetze Zeilenumbrüche
45 straftaten_spalte = 'Straftaten \n-insgesamt-' # Anpassung basierend auf dem
    Beispiel
46
47 # Überprüfe, ob die Spalte existiert
48 if straftaten_spalte not in unterbezirk_df.columns:
49     # Alternativ Zeilenumbrüche entfernen und erneut versuchen
50     unterbezirk_df.columns = unterbezirk_df.columns.str.replace('\n', ' ',
        regex=True)
51     straftaten_spalte = 'Straftaten -insgesamt-'
52     if straftaten_spalte not in unterbezirk_df.columns:
53         raise ValueError(f"Die Spalte '{straftaten_spalte}' wurde nicht
            gefunden.")
54
55 # Gruppiere nach Unterbezirk und summiere die Straftaten
56 aggregierte_straftaten = unterbezirk_df.groupby('Bezeichnung (Bezirksregion)'
    )[straftaten_spalte].sum()
57
58 # 7. Sortiere die Unterbezirke absteigend nach den Straftaten
59 sortierte_straftaten = aggregierte_straftaten.sort_values(ascending=False)

```



```

60
61 # 8. Wähle die Top 10 Unterbezirke mit den höchsten Straftaten
62 top_10_unterbezirke = sortierte_straftaten.head(10)
63
64 # 9. Erstelle einen neuen DataFrame mit den Top 10 Unterbezirken und ihren
    Straftaten
65 final_df = top_10_unterbezirke.reset_index()
66 final_df.columns = ['Bezeichnung (Bezirksregion)', 'Straftaten -insgesamt-']
67
68 # 10. Gib den finalen DataFrame aus
69 print(final_df)
70 '''
71
72 ### Erklärung der Schritte:
73
74 1. **Importieren der notwendigen Bibliothek**: Wir verwenden 'pandas', um
    Excel-Dateien zu verarbeiten.
75
76 2. **Definieren der Oberbezirke**: Die Liste 'oberbezirke' enthält die Namen
    der Oberbezirke, die wir später zur Filterung nutzen.
77
78 3. **Einlesen der Excel-Sheets**: Wir iterieren über die Jahre 2014 bis 2023,
    lesen jedes entsprechende Sheet ein und fügen eine zusätzliche Spalte '
    Jahr' hinzu, um die Daten später ggf. nach Jahr zu analysieren. Alle
    DataFrames werden in eine Liste 'dataframes' gespeichert.
79
80 4. **Kombinieren der DataFrames**: Alle einzelnen Jahres-DataFrames werden
    mit 'pd.concat' zu einem einzigen 'gesamt_df' kombiniert.
81
82 5. **Filtern der Daten**: Wir entfernen alle Zeilen, die zu den Oberbezirken,
    'Berlin (PKS gesamt)' oder nicht zuzuordnende Einträge gehören. Dies
    erfolgt durch die Nutzung der 'isin'-Methode und dem Ausschluss ('~').
83
84 6. **Aggregieren der Straftaten**: Wir gruppieren den gefilterten DataFrame
    nach 'Bezeichnung (Bezirksregion)' und summieren die Werte der Spalte '
    Straftaten -insgesamt-' über alle Jahre hinweg.
85
86 7. **Sortieren der Ergebnisse**: Die aggregierten Straftaten werden
    absteigend sortiert, sodass die Unterbezirke mit den meisten Straftaten
    oben stehen.
87
88 8. **Auswahl der Top 10**: Wir wählen die ersten 10 Einträge aus dem
    sortierten Series-Objekt aus.
89
90 9. **Erstellen des finalen DataFrames**: Der finale DataFrame 'final_df' enth
    ält nur die Unterbezirke und die summierten Straftaten. Die Spalten werden
    entsprechend benannt.
91
92 10. **Ausgabe des Ergebnisses**: Der finale DataFrame wird ausgegeben, sodass
    die Top 10 Unterbezirke mit den höchsten Straftaten angezeigt werden.
93
94 ### Hinweise:

```

```

95
96 - **Spaltennamen**: Es ist wichtig, die exakten Spaltennamen zu verwenden.
    Falls der Spaltenname Zeilenumbrüche enthält ('\n'), kann es notwendig
    sein, die Spaltennamen zu bereinigen oder anzupassen.
97
98 - **Fehlerbehandlung**: Das Skript enthält eine einfache Fehlerbehandlung für
    das Einlesen der Sheets und das Auffinden der notwendigen Spalten.
99
100 - **Anpassungen**: Falls die Struktur der Excel-Datei oder die Spaltennamen
    abweichen, müssen entsprechende Anpassungen vorgenommen werden.
101
102 ### Beispielausgabe:
103
104 Angenommen, die Berechnungen ergeben die folgenden Top 10 Unterbezirke, kö
    nnte die Ausgabe wie folgt aussehen:
105
106 '''
107      Bezeichnung (Bezirksregion)  Straftaten -insgesamt-
108 0          Neukölln                350000
109 1      Friedrichshain-Kreuzberg    300000
110 2          Pankow                  280000
111 3      Charlottenburg-Wilmersdorf  260000
112 4          Lichtenberg             250000
113 5      Tempelhof-Schöneberg        240000
114 6      Treptow-Köpenick            230000
115 7      Marzahn-Hellersdorf         220000
116 8      Steglitz-Zehlendorf         210000
117 9          Reinickendorf           200000
118 '''
119
120 Dies zeigt die Unterbezirke mit den höchsten aggregierten Straftaten über den
    Zeitraum von 2014 bis 2023.

```

Listing 188: API Response Testfall 6 Prompt 3 Ausführung 1

```

1 Hier ist ein Python-Skript, das die von Ihnen beschriebenen Schritte ausführt
  . Das Skript verwendet die 'pandas' Bibliothek, um die Excel-Datei zu
  verarbeiten. Stellen Sie sicher, dass 'pandas' installiert ist ('pip
  install pandas openpyxl'), bevor Sie das Skript ausführen.
2
3 '''python
4 import pandas as pd
5
6 # Schritt 1: Definieren Sie den Dateipfad und die relevanten Sheetnamen
7 excel_datei = 'Fallzahlen&HZ 2014-2023.xlsx'
8 jahre = range(2014, 2024) # 2014 bis 2023 inclusive
9 sheet_namen = [f'Fallzahlen_{jahr}' for jahr in jahre]
10
11 # Schritt 2: Lesen Sie alle Sheets ein und speichern Sie sie in separaten
    DataFrames
12 dataframes = {}
13 for sheet in sheet_namen:
14     try:

```

```

15         df = pd.read_excel(excel_datei, sheet_name=sheet)
16         dataframes[sheet] = df
17         print(f"Sheet '{sheet}' erfolgreich gelesen.")
18     except Exception as e:
19         print(f"Fehler beim Lesen des Sheets '{sheet}': {e}")
20
21 # Prüfen, ob alle Sheets erfolgreich gelesen wurden
22 if len(dataframes) != len(sheet_namen):
23     print("Nicht alle Sheets wurden erfolgreich gelesen. Bitte überprüfen Sie
24         die Sheetnamen und die Excel-Datei.")
25     exit()
26
27 # Schritt 3: Fügen Sie alle DataFrames zu einem einzigen DataFrame zusammen
28 gesamt_df = pd.concat(dataframes.values(), ignore_index=True)
29 print("Alle Sheets wurden erfolgreich zu einem einzigen DataFrame zusammengef
30     ührt.")
31
32 # Schritt 4: Definieren Sie die Oberbezirke
33 oberbezirke = [
34     'Mitte',
35     'Friedrichshain-Kreuzberg',
36     'Pankow',
37     'Charlottenburg-Wilmersdorf',
38     'Spandau',
39     'Steglitz-Zehlendorf',
40     'Tempelhof-Schöneberg',
41     'Neukölln',
42     'Treptow-Köpenick',
43     'Marzahn-Hellersdorf',
44     'Lichtenberg',
45     'Reinickendorf'
46 ]
47
48 # Schritt 5: Definieren Sie die Ausschlusskriterien
49 ausschlusskriterien = oberbezirke + [
50     'Berlin (PKS gesamt)',
51     'Stadt Berlin, nicht zuzuordnen',
52     'Bezirk (Rd), nicht zuzuordnen',
53     'Stadtgebiet Berlin, nicht zuzuordnen'
54 ]
55
56 # Schritt 6: Filtern Sie die DataFrame, um nur Unterbezirke zu behalten
57 unterbezirke_df = gesamt_df[~gesamt_df['Bezeichnung (Bezirksregion)'].isin(
58     ausschlusskriterien)]
59 print(f"Anzahl der verbleibenden Unterbezirke: {unterbezirke_df['Bezeichnung
60     (Bezirksregion)'].nunique()}")
61
62 # Schritt 7: Summieren Sie die Straftaten pro Unterbezirk über alle Jahre
63     hinweg
64 # Stellen Sie sicher, dass die Spalte 'Straftaten \n-insgesamt-' numerisch
65     ist
66 unterbezirke_df['Straftaten \n-insgesamt-'] = pd.to_numeric(unterbezirke_df['

```

```

Straftaten \n-insgesamt-'], errors='coerce')
61
62 # Gruppieren nach 'Bezeichnung (Bezirksregion)' und summieren
63 straftaten_summe = unterbezirke_df.groupby('Bezeichnung (Bezirksregion)')['
Straftaten \n-insgesamt-'].sum().reset_index()
64
65 # Schritt 8: Sortieren Sie die Unterbezirke nach der aggregierten Anzahl an
Straftaten absteigend
66 straftaten_sortiert = straftaten_summe.sort_values(by='Straftaten \n-
insgesamt-', ascending=False)
67
68 # Schritt 9: Wählen Sie die Top 10 Unterbezirke mit den höchsten
Gesamtstraftaten aus
69 top10_unterbezirke = straftaten_sortiert.head(10)
70
71 # Schritt 10: Erstellen Sie den finalen DataFrame mit den gewünschten Spalten
72 finaler_df = top10_unterbezirke[['Bezeichnung (Bezirksregion)', 'Straftaten \
n-insgesamt-']]
73
74 # Schritt 11: Ausgabe des finalen DataFrames
75 print("\nTop 10 Unterbezirke mit den höchsten Gesamtstraftaten:")
76 print(finaler_df.to_string(index=False))
77 '''
78
79 ### Erklärung der Schritte:
80
81 1. **Import der Bibliothek:**
82 - 'pandas' wird für die Datenverarbeitung verwendet.
83
84 2. **Definition des Dateipfads und der Sheetnamen:**
85 - Das Skript erstellt eine Liste von Sheetnamen von 'Fallzahlen_2014' bis
'Fallzahlen_2023'.
86
87 3. **Einlesen der Sheets:**
88 - Jeder Sheet wird einzeln gelesen und in einem Dictionary 'dataframes'
gespeichert. Fehler beim Einlesen werden ausgegeben.
89
90 4. **Zusammenführen der DataFrames:**
91 - Alle einzelnen DataFrames werden zu einem großen DataFrame 'gesamt_df'
kombiniert.
92
93 5. **Definieren der Oberbezirke:**
94 - Eine Liste der Oberbezirke wird erstellt, um diese später auszufiltern.
95
96 6. **Ausschlusskriterien festlegen:**
97 - Neben den Oberbezirken werden weitere Kategorien wie 'Berlin (PKS
gesamt)' oder 'nicht zuzuordnen' hinzugefügt, um unerwünschte Einträge
zu entfernen.
98
99 7. **Filtern der DataFrame:**
100 - Der DataFrame wird so gefiltert, dass nur Unterbezirke übrig bleiben.
101

```

```

102 8. **Summieren der Straftaten:**
103     - Die Spalte 'Straftaten \n-insgesamt-' wird in einen numerischen Typ
104       konvertiert, um Summen bilden zu können.
105     - Die Straftaten werden pro Unterbezirk summiert.
106
107 9. **Sortieren und Auswahl der Top 10:**
108     - Die Unterbezirke werden absteigend nach der Anzahl der Straftaten
109       sortiert.
110     - Die Top 10 werden ausgewählt.
111
112 10. **Erstellen des finalen DataFrames:**
113     - Nur die Spalten 'Bezeichnung (Bezirksregion)' und die aggregierten
114       Straftaten werden ausgewählt.
115
116 ### Zusätzliche Hinweise:
117
118 - **Fehlerbehandlung:** Das Skript überprüft, ob alle erwarteten Sheets
119   erfolgreich eingelesen wurden. Andernfalls wird das Skript beendet und
120   eine Fehlermeldung ausgegeben.
121
122 - **Datenbereinigung:** Die Konvertierung der Straftaten-Spalte in einen
123   numerischen Typ ist wichtig, um sicherzustellen, dass die Summen korrekt
124   berechnet werden. Nicht konvertierbare Werte werden als 'NaN' gesetzt und
125   von der Summe ausgeschlossen.
126
127 - **Optimierung:** Falls die Excel-Datei sehr groß ist, könnte die
128   Performance durch spezifischere Leseoptionen verbessert werden, z.B. durch
129   das Lesen nur relevanter Spalten.
130
131 ### Beispielausgabe:
132
133 '''
134 Sheet 'Fallzahlen_2014' erfolgreich gelesen.
135 ...
136 Sheet 'Fallzahlen_2023' erfolgreich gelesen.
137 Alle Sheets wurden erfolgreich zu einem einzigen DataFrame zusammengeführt.
138 Anzahl der verbleibenden Unterbezirke: 96
139
140 Top 10 Unterbezirke mit den höchsten Gesamtstraftaten:
141 Bezeichnung (Bezirksregion) Straftaten
142
143     Prenzlauer Berg          120000
144     Friedrichshain          115000
145     Kreuzberg                110000
146     Neukölln                105000
147     Lichtenberg             100000
148     Charlottenburg          95000
149     Moabit                   90000
150     Steglitz                 85000
151     Schöneberg              80000

```

```

144         Spandau           75000
145     '''
146
147     *Hinweis:* Die obigen Zahlen sind fiktiv und dienen nur zur Illustration.
148     Ihre tatsächlichen Ergebnisse basieren auf den Daten in Ihrer Excel-Datei.
149
150     ### Ausführung des Skripts:
151
152     1. **Speichern des Skripts:**
153     - Speichern Sie den obigen Code in einer Python-Datei, z.B. '
154       analyse_straftaten.py'.
155
156     2. **Installieren der benötigten Bibliotheken:**
157     '''bash
158     pip install pandas openpyxl
159     '''
160
161     3. **Ausführen des Skripts:**
162     '''bash
163     python analyse_straftaten.py
164     '''
165
166     Das Skript liest die Excel-Datei, verarbeitet die Daten gemäß den angegebenen
167     Schritten und gibt die Top 10 Unterbezirke mit den höchsten
168     Gesamtstrafataten aus.

```

Listing 189: API Response Testfall 6 Prompt 3 Ausführung 2

```

1 Hier ist ein Python-Skript, das die von Ihnen beschriebenen Schritte ausführt
2   . Stellen Sie sicher, dass Sie die benötigten Bibliotheken ('pandas' und '
3   openpyxl') installiert haben. Sie können diese mit 'pip install pandas
4   openpyxl' installieren, falls sie noch nicht vorhanden sind.
5
6   '''python
7   import pandas as pd
8
9   # 1. Alle Sheets einlesen und in separaten DataFrames speichern
10  excel_datei = 'Fallzahlen&HZ 2014-2023.xlsx'
11  sheet_namen = [f'Fallzahlen_{jahr}' for jahr in range(2014, 2024)] # 2014
12                  bis 2023
13
14  # Dictionary zur Speicherung der DataFrames
15  dataframes = {}
16
17  for sheet in sheet_namen:
18      try:
19          df = pd.read_excel(excel_datei, sheet_name=sheet)
20          dataframes[sheet] = df
21      except Exception as e:
22          print(f"Fehler beim Lesen des Sheets {sheet}: {e}")
23
24  # 2. Alle DataFrames zu einem einzigen zusammenführen
25  gesamtdf = pd.concat(dataframes.values(), ignore_index=True)

```

```

22
23 # 3. Oberbezirke definieren
24 oberbezirke = [
25     'Mitte', 'Friedrichshain-Kreuzberg', 'Pankow', 'Charlottenburg-
    Wilmersdorf',
26     'Spandau', 'Steglitz-Zehlendorf', 'Tempelhof-Schöneberg', 'Neukölln',
27     'Treptow-Köpenick', 'Marzahn-Hellersdorf', 'Lichtenberg', 'Reinickendorf'
28 ]
29
30 # 4. Zeilen entfernen, die Oberbezirke, 'Berlin (PKS gesamt)', 'Stadt Berlin,
    nicht zuzuordnen' oder ähnliche enthalten
31 begrenzen_werte = oberbezirke + ['Berlin (PKS gesamt)', 'Stadtgebiet Berlin
    , nicht zuzuordnen', 'Bezirk (Rd), nicht zuzuordnen']
32
33 # Annahme: Spaltenname genau 'Bezeichnung (Bezirksregion)'
34 unterbezirk_df = gesamtdf[~gesamtdf['Bezeichnung (Bezirksregion)'].isin(
    begrenzen_werte)].copy()
35
36 # 5. Straftaten insgesamt über alle Jahre summieren
37 # Beachten Sie den genauen Spaltennamen, eventuell mit Zeilenumbruch und
    Bindestrich
38 # Um den Spaltennamen sicher zu erfassen, können wir nach einer Teilü
    bereinstimmung suchen
39 # Alternativ den exakten Spaltennamen verwenden, wie im Beispiel 'Straftaten
    \n-insgesamt-'
40
41 # Finden des exakten Spaltennamens
42 spaltennamen = unterbezirk_df.columns
43 straftaten_spalte = [spalte for spalte in spaltennamen if 'Straftaten' in
    spalte and 'insgesamt' in spalte]
44 if not straftaten_spalte:
45     raise ValueError("Die Spalte für 'Straftaten insgesamt' wurde nicht
    gefunden.")
46 straftaten_spalte = straftaten_spalte[0]
47
48 # Gruppieren nach Unterbezirk und Summieren der Straftaten
49 aggregiert = unterbezirk_df.groupby('Bezeichnung (Bezirksregion)')[
    straftaten_spalte].sum().reset_index()
50
51 # 6. Sortieren der Unterbezirke absteigend nach Straftaten
52 aggregiert_sortiert = aggregiert.sort_values(by=straftaten_spalte, ascending=
    False)
53
54 # 7. Top 10 Unterbezirke auswählen
55 top_10 = aggregiert_sortiert.head(10)
56
57 # 8. Neuer DataFrame mit den gewünschten Spalten
58 final_df = top_10[['Bezeichnung (Bezirksregion)', straftaten_spalte]].copy()
59
60 # Optional: Spalten umbenennen für Klarheit
61 final_df.rename(columns={
62     'Bezeichnung (Bezirksregion)': 'Unterbezirk',

```

```

63     straftaten_spalte: 'Gesamte Straftaten'
64 }, inplace=True)
65
66 # 9. Finalen DataFrame ausgeben
67 print(final_df)
68 '''
69
70 ### Erklärung der Schritte:
71
72 1. **Sheets einlesen:** Das Skript erstellt eine Liste der erwarteten Sheet-
    Namen von 'Fallzahlen_2014' bis 'Fallzahlen_2023' und liest jedes dieser
    Sheets in einen separaten Pandas DataFrame ein. Diese werden in einem
    Dictionary 'dataframes' gespeichert.
73
74 2. **Zusammenführen der DataFrames:** Alle einzelnen DataFrames werden
    mithilfe von 'pd.concat' zu einem einzigen DataFrame 'gesamtdf'
    zusammengeführt.
75
76 3. **Identifikation der Oberbezirke:** Eine Liste 'oberbezirke' enthält alle
    Oberbezirke, die identifiziert werden sollen.
77
78 4. **Filtern der Daten:** Das Skript entfernt alle Zeilen, die in der Spalte
    'Bezeichnung (Bezirksregion)' entweder einem Oberbezirk, 'Berlin (PKS
    gesamt)', 'Stadtgebiet Berlin, nicht zuzuordnen' oder 'Bezirk (Rd),
    nicht zuzuordnen' entsprechen. Das Ergebnis ist ein DataFrame '
    unterbezirk_df', der nur Unterbezirke enthält.
79
80 5. **Summierung der Straftaten:** Das Skript identifiziert die Spalte für die
    gesamten Straftaten (unter Berücksichtigung möglicher Zeilenumbrüche im
    Spaltennamen) und summiert die Straftaten für jeden Unterbezirk über alle
    Jahre hinweg.
81
82 6. **Sortierung:** Die Unterbezirke werden nach der aggregierten Anzahl der
    Straftaten absteigend sortiert.
83
84 7. **Auswahl der Top 10:** Die zehn Unterbezirke mit den höchsten
    Gesamtstraftaten werden ausgewählt.
85
86 8. **Erstellung des finalen DataFrames:** Ein neuer DataFrame 'final_df' wird
    erstellt, der nur die Spalten 'Unterbezirk' und 'Gesamte Straftaten'
    enthält.
87
88 9. **Ausgabe:** Der finale DataFrame wird ausgegeben.
89
90 ### Hinweis:
91
92 - **Spaltennamen:** Achten Sie darauf, dass die Spaltennamen exakt mit denen
    in Ihrer Excel-Datei übereinstimmen. Insbesondere bei Sonderzeichen wie
    Zeilenumbrüchen ('\n') kann es zu Abweichungen kommen. Das Skript sucht
    nach einer Spalte, die sowohl "Straftaten" als auch "insgesamt" im Namen
    enthält. Passen Sie dies ggf. an Ihre tatsächlichen Spaltennamen an.
93

```



```

94 - **Fehlerbehandlung:** Das Skript enthält grundlegende Fehlerbehandlungen, z
    .B. beim Lesen der Sheets und beim Auffinden der relevanten Spalte.
    Stellen Sie sicher, dass Ihre Excel-Datei korrekt formatiert ist und alle
    erwarteten Sheets und Spalten enthält.
95
96 - **Ausführung:** Stellen Sie sicher, dass sich die Excel-Datei 'Fallzahlen&
    HZ 2014-2023.xlsx' im gleichen Verzeichnis wie das Skript befindet oder
    geben Sie den vollständigen Pfad zur Datei an.
97
98 Ich hoffe, dieses Skript hilft Ihnen weiter! Bei weiteren Fragen stehe ich
    gerne zur Verfügung.

```

Listing 190: API Response Testfall 6 Prompt 3 Ausführung 3

```

1 Hier ist ein Python-Skript, das die von dir beschriebenen Schritte ausführt.
  Das Skript verwendet die Bibliotheken 'pandas' zum Datenhandling und '
  openpyxl' zum Lesen von Excel-Dateien. Stelle sicher, dass diese
  Bibliotheken installiert sind, bevor du das Skript ausführst. 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 # 1. Definiere den Dateinamen und die zu lesenden Sheets
13 excel_file = 'Fallzahlen&HZ 2014-2023.xlsx'
14 jahre = list(range(2014, 2024)) # 2014 bis 2023
15 sheet_names = [f'Fallzahlen_{jahr}' for jahr in jahre]
16
17 # 2. Lese alle Sheets ein und speichere sie in einer Liste von DataFrames
18 dataframes = []
19 for sheet in sheet_names:
20     df = pd.read_excel(excel_file, sheet_name=sheet)
21     dataframes.append(df)
22
23 # 3. Füge alle DataFrames zu einem einheitlichen DataFrame zusammen
24 all_data = pd.concat(dataframes, ignore_index=True)
25
26 # 4. Definiere die Oberbezirke
27 oberbezirke = [
28     'Mitte', 'Friedrichshain-Kreuzberg', 'Pankow', 'Charlottenburg-
29     Wilmerdorf',
30     'Spandau', 'Steglitz-Zehlendorf', 'Tempelhof-Schöneberg', 'Neukölln',
31     'Treptow-Köpenick', 'Marzahn-Hellersdorf', 'Lichtenberg', 'Reinickendorf'
32 ]
33
34 # Zusätzliche nicht zuzuordnende Einträge
35 nicht_zuordenbar = [

```

```

35     'Berlin (PKS gesamt)',
36     'Bezirk (Rd), nicht zuzuordnen',
37     'Stadtgebiet Berlin, nicht zuzuordnen'
38 ]
39
40 # 5. Entferne alle Zeilen, die Oberbezirke oder nicht zuordenbare Einträge
    enthalten
41 filtered_data = all_data[~all_data['Bezeichnung (Bezirksregion)'].isin(
    oberbezirke + nicht_zuordenbar)]
42
43 # 6. Entferne weitere allgemeine Einträge, falls vorhanden (optional)
44 # Falls es noch andere allgemeine Einträge gibt, können diese hier hinzugefü
    gt werden
45 # Beispielsweise: 'Gesamt Berlin' oder ähnliche
46
47 # 7. Aggregiere die Straftaten pro Unterbezirk über alle Jahre hinweg
48 # Zuerst sicherstellen, dass die Spalte korrekt benannt ist
49 # Manchmal können Leerzeichen oder Zeilenumbrüche Probleme verursachen
50
51 # Liste der möglichen Spaltennamen für 'Straftaten -insgesamt-'
52 straftaten_spalten = [
53     'Straftaten \n-insgesamt-',
54     'Straftaten -insgesamt-',
55     'Straftaten - insgesamt -' # Falls es Varianten gibt
56 ]
57
58 # Finde die existierende Spalte
59 straftaten_spalte = None
60 for col in straftaten_spalten:
61     if col in filtered_data.columns:
62         straftaten_spalte = col
63         break
64
65 if not straftaten_spalte:
66     raise ValueError("Die Spalte für 'Straftaten insgesamt' wurde nicht
    gefunden.")
67
68 # Gruppiere nach 'Bezeichnung (Bezirksregion)' und summiere die Straftaten
69 aggregated = filtered_data.groupby('Bezeichnung (Bezirksregion)')[
    straftaten_spalte].sum().reset_index()
70
71 # 8. Sortiere die Unterbezirke absteigend nach der aggregierten Anzahl an
    Straftaten
72 aggregated_sorted = aggregated.sort_values(by=straftaten_spalte, ascending=
    False)
73
74 # 9. Wähle die Top 10 Unterbezirke mit den höchsten Gesamtstraftaten aus
75 top_10 = aggregated_sorted.head(10)
76
77 # 10. Erstelle den finalen DataFrame mit den gewünschten Spalten
78 final_df = top_10[['Bezeichnung (Bezirksregion)', straftaten_spalte]].copy()
79 final_df = final_df.rename(columns={

```

```

80     'Bezeichnung (Bezirksregion)': 'Unterbezirk',
81     straftaten_spalte: 'Gesamtstraftaten'
82 })
83
84 # 11. Ausgabe des finalen DataFrames
85 print(final_df)
86 '''
87
88 ### Erklärung der Schritte:
89
90 1. **Datei und Sheets definieren**: Das Skript definiert den Dateinamen und
    erstellt eine Liste der relevanten Sheet-Namen von 2014 bis 2023.
91
92 2. **Sheets einlesen**: Es liest jedes Sheet einzeln ein und speichert die
    resultierenden DataFrames in einer Liste.
93
94 3. **DataFrames zusammenführen**: Alle einzelnen DataFrames werden zu einem
    großen DataFrame zusammengeführt, der alle Jahre umfasst.
95
96 4. **Oberbezirke und nicht zuordenbare Einträge definieren**: Eine Liste der
    Oberbezirke sowie weiterer Einträge, die ausgeschlossen werden sollen,
    wird erstellt.
97
98 5. **Filtern der Daten**: Das Skript entfernt alle Zeilen, die zu den
    Oberbezirken oder den nicht zuordenbaren Einträgen gehören.
99
100 6. **Optionales Entfernen weiterer Einträge**: Falls es weitere allgemeine
    Einträge gibt, können diese hier hinzugefügt und entfernt werden.
101
102 7. **Aggregieren der Straftaten**: Die Straftaten werden pro Unterbezirk über
    alle Jahre hinweg summiert. Dabei wird sichergestellt, dass die richtige
    Spalte verwendet wird, auch wenn sie unterschiedliche Namen aufgrund von
    Leerzeichen oder Zeilenumbrüchen haben könnte.
103
104 8. **Sortieren**: Die Unterbezirke werden absteigend nach der Gesamtanzahl
    der Straftaten sortiert.
105
106 9. **Top 10 auswählen**: Die zehn Unterbezirke mit den höchsten
    Gesamtstraftaten werden ausgewählt.
107
108 10. **Finalen DataFrame erstellen**: Ein neuer DataFrame wird erstellt, der
    nur die Unterbezirke und ihre aggregierten Straftaten enthält und
    entsprechend umbenannt wird.
109
110 11. **Ausgabe**: Der finale DataFrame wird ausgegeben.
111
112 ### Beispielhafte Ausgabe:
113
114 '''
115
116         Unterbezirk  Gesamtstraftaten
117 0      ...
118 1      ...

```

```

118 ...
119 9 ...
120 '''
121
122 Das Skript gibt die Top 10 Unterbezirke mit den höchsten Gesamtstraftaten ü
ber den Zeitraum von 2014 bis 2023 aus. Stelle sicher, dass die
Spaltennamen exakt mit denen in deiner Excel-Datei übereinstimmen. Falls
die Spalte für die Straftaten leicht abweicht, kannst du die Liste '
straftaten_spalten' entsprechend anpassen.

```

Listing 191: API Response Testfall 6 Prompt 3 Ausführung 4

```

1 Hier ist ein vollständiges Python-Skript, das die von dir beschriebenen
  Schritte ausführt. Das Skript verwendet die Bibliothek 'pandas', um die
  Excel-Datei zu verarbeiten. Stelle sicher, dass du 'pandas' und 'openpyxl'
  installiert hast, bevor du das Skript ausführst. Du kannst sie mit 'pip'
  installieren, falls noch nicht geschehen:
2
3 '''bash
4 pip install pandas openpyxl
5 '''
6
7 ### Python-Skript
8
9 '''python
10 import pandas as pd
11
12 # 1. Definiere den Pfad zur Excel-Datei
13 excel_datei = 'Fallzahlen&HZ 2014-2023.xlsx'
14
15 # 2. Erstelle eine Liste der Sheet-Namen von 'Fallzahlen_2014' bis '
    Fallzahlen_2023'
16 jahre = range(2014, 2024) # 2024 ist exklusiv, also bis 2023
17 sheet_namen = [f'Fallzahlen_{jahr}' for jahr in jahre]
18
19 # 3. Lese alle Sheets ein und speichere jedes in einem separaten DataFrame
20 dataframes = []
21 for sheet in sheet_namen:
22     try:
23         df = pd.read_excel(excel_datei, sheet_name=sheet)
24         dataframes.append(df)
25         print(f"Sheet '{sheet}' erfolgreich eingelesen.")
26     except Exception as e:
27         print(f"Fehler beim Einlesen von Sheet '{sheet}': {e}")
28
29 # 4. Füge alle DataFrames zu einem einzigen zusammen
30 kombinierter_df = pd.concat(dataframes, ignore_index=True)
31 print("Alle Sheets erfolgreich zu einem DataFrame kombiniert.")
32
33 # 5. Definiere die Liste der Oberbezirke
34 oberbezirke = [
35     'Mitte',
36     'Friedrichshain-Kreuzberg',

```

```

37     'Pankow',
38     'Charlottenburg-Wilmersdorf',
39     'Spandau',
40     'Steglitz-Zehlendorf',
41     'Tempelhof-Schöneberg',
42     'Neukölln',
43     'Treptow-Köpenick',
44     'Marzahn-Hellersdorf',
45     'Lichtenberg',
46     'Reinickendorf'
47 ]
48
49 # 6. Definiere zusätzliche auszuschließende Einträge
50 ausschluss_liste = oberbezirke + [
51     'Berlin (PKS gesamt)',
52     'Stadtgebiet Berlin, nicht zuzuordnen',
53     'Bezirk (Rd), nicht zuzuordnen'
54 ]
55
56 # 7. Entferne alle Zeilen, die in der Ausschlussliste enthalten sind
57 unterbezirke_df = kombinierter_df[~kombinierter_df['Bezeichnung (
    Bezirksregion)'].isin(ausschluss_liste)]
58 print(f"Anzahl der verbleibenden Unterbezirke: {unterbezirke_df['Bezeichnung
    (Bezirksregion)'].nunique()}")
59
60 # 8. Stelle sicher, dass die Spalte 'Straftaten \n-insgesamt-' numerisch ist
61 unterbezirke_df['Straftaten \n-insgesamt-'] = pd.to_numeric(unterbezirke_df['
    Straftaten \n-insgesamt-'], errors='coerce')
62
63 # 9. Entferne mögliche NaN-Werte in der Straftaten-Spalte
64 unterbezirke_df = unterbezirke_df.dropna(subset=['Straftaten \n-insgesamt-'])
65
66 # 10. Summiere die Straftaten pro Unterbezirk über alle Jahre hinweg
67 aggregierte_straftaten = unterbezirke_df.groupby('Bezeichnung (Bezirksregion)
    ')[ 'Straftaten \n-insgesamt-'].sum()
68
69 # 11. Sortiere die Unterbezirke absteigend nach der aggregierten Anzahl an
    Straftaten
70 sortierte_straftaten = aggregierte_straftaten.sort_values(ascending=False)
71
72 # 12. Wähle die Top 10 Unterbezirke mit den höchsten Gesamtstraftaten aus
73 top_10_unterbezirke = sortierte_straftaten.head(10)
74
75 # 13. Erstelle einen neuen DataFrame mit den gewünschten Spalten
76 finaler_df = top_10_unterbezirke.reset_index()
77 finaler_df.columns = ['Bezeichnung (Bezirksregion)', 'Straftaten -insgesamt-'
    ]
78
79 # 14. Gib den finalen DataFrame aus
80 print("\nTop 10 Unterbezirke mit den höchsten Gesamtstraftaten:")
81 print(finaler_df)
82 '''

```

```

83
84 ### Erklärung der einzelnen Schritte:
85
86 1. **Importieren der Bibliothek**: 'pandas' wird für die Datenverarbeitung
    verwendet.
87
88 2. **Definieren des Excel-Dateipfads**: Stelle sicher, dass sich die Datei '
    Fallzahlen&HZ 2014-2023.xlsx' im gleichen Verzeichnis wie das Skript
    befindet oder gib den vollständigen Pfad an.
89
90 3. **Erstellen der Sheet-Namen**: Die Sheets werden anhand der Jahre von 2014
    bis 2023 benannt.
91
92 4. **Einlesen der Sheets**: Für jedes Jahr wird das entsprechende Sheet
    eingelesen und in einer Liste gespeichert. Fehler beim Einlesen einzelner
    Sheets werden angezeigt, das Skript fährt aber fort.
93
94 5. **Kombinieren der DataFrames**: Alle einzelnen DataFrames werden zu einem
    großen DataFrame zusammengeführt.
95
96 6. **Definieren der Oberbezirke**: Eine Liste der Oberbezirke wird erstellt.
97
98 7. **Definieren zusätzlicher Ausschlüsse**: Neben den Oberbezirken gibt es
    weitere Einträge, die entfernt werden sollen.
99
100 8. **Filtern der Unterbezirke**: Alle Zeilen, die in der Ausschlussliste
    enthalten sind, werden entfernt, sodass nur Unterbezirke verbleiben.
101
102 9. **Sicherstellen der numerischen Werte**: Die Spalte für Straftaten wird in
    numerische Werte umgewandelt. Fehlerhafte Einträge werden als 'NaN'
    markiert.
103
104 10. **Entfernen von NaN-Werten**: Alle Zeilen mit fehlenden Straftaten werden
    entfernt.
105
106 11. **Aggregieren der Straftaten**: Die Straftaten werden pro Unterbezirk
    summiert.
107
108 12. **Sortieren der Ergebnisse**: Die Unterbezirke werden absteigend nach der
    Anzahl der Straftaten sortiert.
109
110 13. **Auswählen der Top 10**: Die zehn Unterbezirke mit den höchsten
    Straftaten werden ausgewählt.
111
112 14. **Erstellen und Anzeigen des finalen DataFrames**: Der finale DataFrame
    mit den gewünschten Spalten wird erstellt und ausgegeben.
113
114 ### Beispielausgabe
115
116 Nach dem Ausführen des Skripts erhältst du eine Ausgabe ähnlich der folgenden
    (die tatsächlichen Werte hängen von den Daten ab):
117

```

```

118 '''
119 Top 10 Unterbezirke mit den höchsten Gesamtstraftaten:
120     Bezeichnung (Bezirksregion)  Straftaten -insgesamt-
121 0          Neukölln                123456
122 1          Tempelhof-Schöneberg    112233
123 2          ...                    ...
124 '''
125
126 Dieses Skript erfüllt alle von dir genannten Anforderungen und sollte dir
    eine klare Übersicht über die Unterbezirke mit den höchsten
    Gesamtstraftaten bieten.

```

Listing 192: API Response Testfall 6 Prompt 3 Ausführung 5

11.3.14 Komplette Ausgaben der Skripte Testfall 1

```

1 Original DataFrame:
2   LOR-Schlüssel (Bezirksregion)                Bezirke
   Straftaten_insgesamt Raub ... Sachbeschädigung_insgesamt Sachbeschä
   digung_Graffiti Rauschgiftdelikte Kieztaten
3 0                                10000                Mitte
   83611    884    ...                    6172                976
   3306    19212
4 1                                20000    Friedrichshain-Kreuzberg
   55838    923    ...                    4906                801
   4455    12724
5 2                                30000                Pankow
   39164    529    ...                    4576                942
   910    10043
6 3                                40000    Charlottenburg-Wilmersdorf
   53630    464    ...                    4134                750
   1431    12175
7 4                                50000                Spandau
   25564    247    ...                    2379                226
   645    6473
8
9 [5 rows x 19 columns]
10
11 Sortierter DataFrame nach 'Straftaten_insgesamt':
12   LOR-Schlüssel (Bezirksregion)                Bezirke
   Straftaten_insgesamt Raub ... Sachbeschädigung_insgesamt Sachbeschä
   digung_Graffiti Rauschgiftdelikte Kieztaten
13 0                                999999                Berlin (PKS gesamt)
   536695    5389    ...                    45434
   7448    17502    119737
14 1                                10000                Mitte
   83611    884    ...                    6172
   976    3306    19212
15 2                                999900    Stadtgebiet Berlin, nicht zuzuordnen
   56425    98    ...                    1065
   490    734    2558

```

16	3			20000		Friedrichshain-Kreuzberg
		55838	923	...		4906
		801		4455	12724	
17	4			40000		Charlottenburg-Wilmersdorf
		53630	464	...		4134
		750		1431	12175	
18	5			80000		Neukölln
		45313	508	...		4165
		379		1869	10253	
19	6			70000		Tempelhof-Schöneberg
		41776	328	...		3592
		622		1435	10132	
20	7			30000		Pankow
		39164	529	...		4576
		942		910	10043	
21	8			110000		Lichtenberg
		31563	362	...		3206
		628		663	8499	
22	9			120000		Reinickendorf
		27344	311	...		2481
		203		553	7323	
23	10			100000		Marzahn-Hellersdorf
		26649	315	...		2908
		392		424	8203	
24	11			90000		Treptow-Köpenick
		26099	207	...		3152
		558		680	6720	
25	12			50000		Spandau
		25564	247	...		2379
		226		645	6473	
26	13			60000		Steglitz-Zehlendorf
		23719	213	...		2698
		481		397	5422	

[14 rows x 19 columns]

Der sortierte DataFrame wurde in 'Fallzahlen_sortiert.xlsx' gespeichert.

Command being timed: "python testcase1/prompt1/exec1/script.py"

User time (seconds): 2.38

System time (seconds): 0.05

Percent of CPU this job got: 476%

Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.51

Average shared text size (kbytes): 0

Average unshared data size (kbytes): 0

Average stack size (kbytes): 0

Average total size (kbytes): 0

Maximum resident set size (kbytes): 150412

Average resident set size (kbytes): 0

Major (requiring I/O) page faults: 0

Minor (reclaiming a frame) page faults: 32490

Voluntary context switches: 36

Involuntary context switches: 116


```

46 Swaps: 0
47 File system inputs: 0
48 File system outputs: 80
49 Socket messages sent: 0
50 Socket messages received: 0
51 Signals delivered: 0
52 Page size (bytes): 4096
53 Exit status: 0

```

Listing 193: Ausgabe Testfall 1Prompt 1Ausführung 1

```

1 Unsortierte Daten:
2   LOR-Schlüssel (Bezirksregion)                               Bezirke
   Straftaten_insgesamt Raub ... Sachbeschädigung_insgesamt Sachbeschä
   digung_Graffiti Rauschgiftdelikte Kieztaten
3 0                               10000                               Mitte
   83611   884   ...                               6172                               976
   3306   19212
4 1                               20000   Friedrichshain-Kreuzberg
   55838   923   ...                               4906                               801
   4455   12724
5 2                               30000                               Pankow
   39164   529   ...                               4576                               942
   910   10043
6 3                               40000   Charlottenburg-Wilmersdorf
   53630   464   ...                               4134                               750
   1431   12175
7 4                               50000                               Spandau
   25564   247   ...                               2379                               226
   645   6473
8
9 [5 rows x 19 columns]
10
11 Sortierte Daten nach 'Straftaten_insgesamt':
12   LOR-Schlüssel (Bezirksregion)                               Bezirke
   Straftaten_insgesamt Raub ... Sachbeschädigung_insgesamt Sachbeschä
   digung_Graffiti Rauschgiftdelikte Kieztaten
13 13                               999999                               Berlin (PKS gesamt)
   536695   5389   ...                               45434
   7448                               17502   119737
14 0                               10000                               Mitte
   83611   884   ...                               6172
   976                               3306   19212
15 12                               999900   Stadtgebiet Berlin, nicht zuzuordnen
   56425   98   ...                               1065
   490                               734   2558
16 1                               20000                               Friedrichshain-Kreuzberg
   55838   923   ...                               4906
   801                               4455   12724
17 3                               40000                               Charlottenburg-Wilmersdorf
   53630   464   ...                               4134
   750                               1431   12175
18

```

```

19 [5 rows x 19 columns]
20
21 Die sortierten Daten wurden in 'Fallzahlen_sortiert.xlsx' gespeichert.
22 Command being timed: "python testcase1/prompt1/exec2/script.py"
23 User time (seconds): 2.39
24 System time (seconds): 0.06
25 Percent of CPU this job got: 496%
26 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.49
27 Average shared text size (kbytes): 0
28 Average unshared data size (kbytes): 0
29 Average stack size (kbytes): 0
30 Average total size (kbytes): 0
31 Maximum resident set size (kbytes): 150784
32 Average resident set size (kbytes): 0
33 Major (requiring I/O) page faults: 0
34 Minor (reclaiming a frame) page faults: 32519
35 Voluntary context switches: 46
36 Involuntary context switches: 253
37 Swaps: 0
38 File system inputs: 0
39 File system outputs: 176
40 Socket messages sent: 0
41 Socket messages received: 0
42 Signals delivered: 0
43 Page size (bytes): 4096
44 Exit status: 0

```

Listing 194: Ausgabe Testfall 1Prompt 1Ausführung 2

```

1 Excel-Datei erfolgreich eingelesen.
2 DataFrame erfolgreich nach 'Straftaten_insgesamt' sortiert.
3   LOR-Schlüssel (Bezirksregion)                                Bezirke
   Straftaten_insgesamt Raub ... Sachbeschädigung_insgesamt Sachbeschä
   digung_Graffiti Rauschgiftdelikte Kieztaten
4 0
   536695 5389 ... 45434
   7448      17502 119737
5 1
   83611 884 ... 6172
   976      3306 19212
6 2
   56425 98 ... 1065
   490      734 2558
7 3
   55838 923 ... 4906
   801      4455 12724
8 4
   53630 464 ... 4134
   750      1431 12175
9 5
   45313 508 ... 4165
   379      1869 10253
10 6
   70000
   Tempelhof-Schöneberg

```

```

11 7      41776  328  ...      3592
      622      1435      10132
      39164  529  ...      4576
      942      910      10043
12 8      110000
      31563  362  ...      3206
      628      663      8499
13 9      120000
      27344  311  ...      2481
      203      553      7323
14 10     100000
      26649  315  ...      2908
      392      424      8203
15 11     90000
      26099  207  ...      3152
      558      680      6720
16 12     50000
      25564  247  ...      2379
      226      645      6473
17 13     60000
      23719  213  ...      2698
      481      397      5422

```

```

18
19 [14 rows x 19 columns]
20 Command being timed: "python testcase1/prompt1/exec3/script.py"
21 User time (seconds): 2.37
22 System time (seconds): 0.06
23 Percent of CPU this job got: 474%
24 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.51
25 Average shared text size (kbytes): 0
26 Average unshared data size (kbytes): 0
27 Average stack size (kbytes): 0
28 Average total size (kbytes): 0
29 Maximum resident set size (kbytes): 150900
30 Average resident set size (kbytes): 0
31 Major (requiring I/O) page faults: 2
32 Minor (reclaiming a frame) page faults: 32483
33 Voluntary context switches: 36
34 Involuntary context switches: 85
35 Swaps: 0
36 File system inputs: 128
37 File system outputs: 0
38 Socket messages sent: 0
39 Socket messages received: 0
40 Signals delivered: 0
41 Page size (bytes): 4096
42 Exit status: 0

```

Listing 195: Ausgabe Testfall 1Prompt 1Ausführung 3

1	LOR-Schlüssel (Bezirksregion)	Bezirke
	Straftaten_insgesamt Raub ... Sachbeschädigung_insgesamt Sachbeschä	

	digung_Graffiti	Rauschgiftdelikte	Kieztaten	
2	13	999999	Berlin (PKS gesamt)	
		536695 5389 ...	45434	
		7448	17502 119737	
3	0	10000	Mitte	
		83611 884 ...	6172	
		976	3306 19212	
4	12	999900	Stadtgebiet Berlin, nicht zuzuordnen	
		56425 98 ...	1065	
		490	734 2558	
5	1	20000	Friedrichshain-Kreuzberg	
		55838 923 ...	4906	
		801	4455 12724	
6	3	40000	Charlottenburg-Wilmersdorf	
		53630 464 ...	4134	
		750	1431 12175	
7	7	80000	Neukölln	
		45313 508 ...	4165	
		379	1869 10253	
8	6	70000	Tempelhof-Schöneberg	
		41776 328 ...	3592	
		622	1435 10132	
9	2	30000	Pankow	
		39164 529 ...	4576	
		942	910 10043	
10	10	110000	Lichtenberg	
		31563 362 ...	3206	
		628	663 8499	
11	11	120000	Reinickendorf	
		27344 311 ...	2481	
		203	553 7323	
12	9	100000	Marzahn-Hellersdorf	
		26649 315 ...	2908	
		392	424 8203	
13	8	90000	Treptow-Köpenick	
		26099 207 ...	3152	
		558	680 6720	
14	4	50000	Spandau	
		25564 247 ...	2379	
		226	645 6473	
15	5	60000	Steglitz-Zehlendorf	
		23719 213 ...	2698	
		481	397 5422	

```

16
17 [14 rows x 19 columns]
18 Command being timed: "python testcase1/prompt1/exec4/script.py"
19 User time (seconds): 2.40
20 System time (seconds): 0.05
21 Percent of CPU this job got: 481%
22 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.51
23 Average shared text size (kbytes): 0
24 Average unshared data size (kbytes): 0

```

```

25 Average stack size (kbytes): 0
26 Average total size (kbytes): 0
27 Maximum resident set size (kbytes): 153324
28 Average resident set size (kbytes): 0
29 Major (requiring I/O) page faults: 0
30 Minor (reclaiming a frame) page faults: 32484
31 Voluntary context switches: 34
32 Involuntary context switches: 349
33 Swaps: 0
34 File system inputs: 0
35 File system outputs: 176
36 Socket messages sent: 0
37 Socket messages received: 0
38 Signals delivered: 0
39 Page size (bytes): 4096
40 Exit status: 0

```

Listing 196: Ausgabe Testfall 1Prompt 1Ausführung 4

```

1 Daten erfolgreich eingelesen.
2 Verfügbare Spalten:
3 ['LOR-Schlüssel (Bezirksregion)', 'Bezirke', 'Straftaten_insgesamt', 'Raub',
  'Straßenraub', 'Körperverletzungen_insgesamt', 'schwere_ Körperverletzung',
  'Freiheitsberaubung_Nötigung', 'Diebstahl_insgesamt', '
  Diebstahl_Kraftwagen', 'Diebstahl_Kfz', 'Fahrraddiebstahl', '
  Wohnraumeinbruch', 'Branddelikte_insgesamt', 'Brandstiftung', 'Sachbeschä
  digung_insgesamt', 'Sachbeschädigung_Graffiti', 'Rauschgiftdelikte', '
  Kieztaten']
4 Daten erfolgreich sortiert.
5 Sortiertes DataFrame:
6   LOR-Schlüssel (Bezirksregion)      Bezirke
   Straftaten_insgesamt Raub ... Sachbeschädigung_insgesamt Sachbeschä
   digung_Graffiti  Rauschgiftdelikte Kieztaten
7 0                                999999      Berlin (PKS gesamt)
      536695  5389 ...                                45434
      7448                                17502  119737
8 1                                10000                                Mitte
      83611  884 ...                                6172
      976                                3306  19212
9 2                                999900 Stadtgebiet Berlin, nicht zuzuordnen
      56425  98 ...                                1065
      490                                734  2558
10 3                                20000      Friedrichshain-Kreuzberg
      55838  923 ...                                4906
      801                                4455  12724
11 4                                40000      Charlottenburg-Wilmersdorf
      53630  464 ...                                4134
      750                                1431  12175
12 5                                80000                                Neukölln
      45313  508 ...                                4165
      379                                1869  10253
13 6                                70000      Tempelhof-Schöneberg
      41776  328 ...                                3592

```

```

14 7      622      1435      10132
      30000      Pankow
      39164      529      ...      4576
      942      910      10043
15 8      110000      Lichtenberg
      31563      362      ...      3206
      628      663      8499
16 9      120000      Reinickendorf
      27344      311      ...      2481
      203      553      7323
17 10     100000      Marzahn-Hellersdorf
      26649      315      ...      2908
      392      424      8203
18 11     90000      Treptow-Köpenick
      26099      207      ...      3152
      558      680      6720
19 12     50000      Spandau
      25564      247      ...      2379
      226      645      6473
20 13     60000      Steglitz-Zehlendorf
      23719      213      ...      2698
      481      397      5422
21
22 [14 rows x 19 columns]
23 Command being timed: "python testcase1/prompt1/exec5/script.py"
24 User time (seconds): 2.36
25 System time (seconds): 0.06
26 Percent of CPU this job got: 478%
27 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.50
28 Average shared text size (kbytes): 0
29 Average unshared data size (kbytes): 0
30 Average stack size (kbytes): 0
31 Average total size (kbytes): 0
32 Maximum resident set size (kbytes): 153112
33 Average resident set size (kbytes): 0
34 Major (requiring I/O) page faults: 0
35 Minor (reclaiming a frame) page faults: 32480
36 Voluntary context switches: 34
37 Involuntary context switches: 269
38 Swaps: 0
39 File system inputs: 0
40 File system outputs: 0
41 Socket messages sent: 0
42 Socket messages received: 0
43 Signals delivered: 0
44 Page size (bytes): 4096
45 Exit status: 0

```

Listing 197: Ausgabe Testfall 1Prompt 1Ausführung 5

```

1 Originaldaten:
2   LOR-Schlüssel      Bezirke  Straftaten_insgesamt  Raub
   Straßenraub ... Brandstiftung  Sachbeschädigung_insgesamt  Sachbeschä

```

		digung_Graffiti	Rauschgiftdelikte	Kieztaten		
3	0	10000		Mitte	83611	884
		521 ...	137		6172	
		976	3306	19212		
4	1	20000	Friedrichshain-Kreuzberg		55838	923
		606 ...	85		4906	
		801	4455	12724		
5	2	30000		Pankow	39164	529
		389 ...	99		4576	
		942	910	10043		
6	3	40000	Charlottenburg-Wilmersdorf		53630	464
		278 ...	79		4134	
		750	1431	12175		
7	4	50000		Spandau	25564	247
		144 ...	111		2379	
		226	645	6473		

8
9 [5 rows x 19 columns]

10
11 Gefilterte Daten:

		LOR-Schlüssel	Bezirke	Straftaten_insgesamt	Raub
		Straßenraub ...	Brandstiftung	Sachbeschädigung_insgesamt	Sachbeschä
		digung_Graffiti	Rauschgiftdelikte	Kieztaten	
13	0	10000		Mitte	83611 884
		521 ...	137		6172
		976	3306	19212	
14	1	20000	Friedrichshain-Kreuzberg		55838 923
		606 ...	85		4906
		801	4455	12724	
15	2	30000		Pankow	39164 529
		389 ...	99		4576
		942	910	10043	
16	3	40000	Charlottenburg-Wilmersdorf		53630 464
		278 ...	79		4134
		750	1431	12175	
17	4	50000		Spandau	25564 247
		144 ...	111		2379
		226	645	6473	

18
19 [5 rows x 19 columns]

20
21 Sortierte Daten:

		LOR-Schlüssel	Bezirke	Straftaten_insgesamt	Raub
		Straßenraub ...	Brandstiftung	Sachbeschädigung_insgesamt	Sachbeschä
		digung_Graffiti	Rauschgiftdelikte	Kieztaten	
23	0	10000		Mitte	83611 884
		521 ...	137		6172
		976	3306	19212	
24	1	20000	Friedrichshain-Kreuzberg		55838 923
		606 ...	85		4906
		801	4455	12724	
25	3	40000	Charlottenburg-Wilmersdorf		53630 464

		278 ...	79		4134	
		750	1431	12175		
26	7	80000		Neukölln	45313	508
		294 ...	145		4165	
		379	1869	10253		
27	6	70000		Tempelhof-Schöneberg	41776	328
		198 ...	86		3592	
		622	1435	10132		
28	2	30000		Pankow	39164	529
		389 ...	99		4576	
		942	910	10043		
29	10	110000		Lichtenberg	31563	362
		259 ...	79		3206	
		628	663	8499		
30	11	120000		Reinickendorf	27344	311
		173 ...	76		2481	
		203	553	7323		
31	9	100000		Marzahn-Hellersdorf	26649	315
		177 ...	96		2908	
		392	424	8203		
32	8	90000		Treptow-Köpenick	26099	207
		118 ...	77		3152	
		558	680	6720		
33	4	50000		Spandau	25564	247
		144 ...	111		2379	
		226	645	6473		
34	5	60000		Steglitz-Zehlendorf	23719	213
		129 ...	52		2698	
		481	397	5422		

```

35
36 [12 rows x 19 columns]
37 Command being timed: "python testcase1/prompt2/exec1/script.py"
38 User time (seconds): 2.37
39 System time (seconds): 0.06
40 Percent of CPU this job got: 492%
41 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.49
42 Average shared text size (kbytes): 0
43 Average unshared data size (kbytes): 0
44 Average stack size (kbytes): 0
45 Average total size (kbytes): 0
46 Maximum resident set size (kbytes): 152296
47 Average resident set size (kbytes): 0
48 Major (requiring I/O) page faults: 0
49 Minor (reclaiming a frame) page faults: 32526
50 Voluntary context switches: 35
51 Involuntary context switches: 226
52 Swaps: 0
53 File system inputs: 0
54 File system outputs: 0
55 Socket messages sent: 0
56 Socket messages received: 0
57 Signals delivered: 0

```



```
58 Page size (bytes): 4096
59 Exit status: 0
```

Listing 198: Ausgabe Testfall 1Prompt 2Ausführung 1

```
1 Excel-Datei erfolgreich eingelesen.
2 Sortierte Daten nach 'Straftaten_insgesamt':
3   LOR-Schlüssel           Bezirke  Straftaten_insgesamt  Raub
   Straßenraub ... Brandstiftung Sachbeschädigung_insgesamt Sachbeschä
   digung_Graffiti Rauschgiftdelikte Kieztaten
4 0      10000           Mitte           83611    884
   521 ...           137           6172
   976           3306    19212
5 1      20000 Friedrichshain-Kreuzberg           55838    923
   606 ...           85           4906
   801           4455    12724
6 2      40000 Charlottenburg-Wilmersdorf           53630    464
   278 ...           79           4134
   750           1431    12175
7 3      80000           Neukölln           45313    508
   294 ...           145           4165
   379           1869    10253
8 4      70000 Tempelhof-Schöneberg           41776    328
   198 ...           86           3592
   622           1435    10132
9 5      30000           Pankow           39164    529
   389 ...           99           4576
   942           910    10043
10 6     110000           Lichtenberg           31563    362
   259 ...           79           3206
   628           663    8499
11 7     120000           Reinickendorf           27344    311
   173 ...           76           2481
   203           553    7323
12 8     100000 Marzahn-Hellersdorf           26649    315
   177 ...           96           2908
   392           424    8203
13 9     90000 Treptow-Köpenick           26099    207
   118 ...           77           3152
   558           680    6720
14 10     50000           Spandau           25564    247
   144 ...           111           2379
   226           645    6473
15 11     60000 Steglitz-Zehlendorf           23719    213
   129 ...           52           2698
   481           397    5422
16
17 [12 rows x 19 columns]
18 Command being timed: "python testcase1/prompt2/exec2/script.py"
19 User time (seconds): 2.38
20 System time (seconds): 0.06
21 Percent of CPU this job got: 480%
22 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.50
```

```

23 Average shared text size (kbytes): 0
24 Average unshared data size (kbytes): 0
25 Average stack size (kbytes): 0
26 Average total size (kbytes): 0
27 Maximum resident set size (kbytes): 153080
28 Average resident set size (kbytes): 0
29 Major (requiring I/O) page faults: 0
30 Minor (reclaiming a frame) page faults: 32477
31 Voluntary context switches: 34
32 Involuntary context switches: 384
33 Swaps: 0
34 File system inputs: 0
35 File system outputs: 0
36 Socket messages sent: 0
37 Socket messages received: 0
38 Signals delivered: 0
39 Page size (bytes): 4096
40 Exit status: 0

```

Listing 199: Ausgabe Testfall 1 Prompt 2 Ausführung 2

	LOR-Schlüssel	Bezirke	Straftaten_insgesamt	Raub	
	Straßenraub ... Brandstiftung Sachbeschädigung_insgesamt Sachbeschädigung_Graffiti Rauschgiftdelikte Kieztaten				
2	0	10000	Mitte	83611	884
	521 ...	137	6172		
	976	3306	19212		
3	1	20000	Friedrichshain-Kreuzberg	55838	923
	606 ...	85	4906		
	801	4455	12724		
4	2	40000	Charlottenburg-Wilmersdorf	53630	464
	278 ...	79	4134		
	750	1431	12175		
5	3	80000	Neukölln	45313	508
	294 ...	145	4165		
	379	1869	10253		
6	4	70000	Tempelhof-Schöneberg	41776	328
	198 ...	86	3592		
	622	1435	10132		
7	5	30000	Pankow	39164	529
	389 ...	99	4576		
	942	910	10043		
8	6	110000	Lichtenberg	31563	362
	259 ...	79	3206		
	628	663	8499		
9	7	120000	Reinickendorf	27344	311
	173 ...	76	2481		
	203	553	7323		
10	8	100000	Marzahn-Hellersdorf	26649	315
	177 ...	96	2908		
	392	424	8203		
11	9	90000	Treptow-Köpenick	26099	207
	118 ...	77	3152		

```

12      558      680      6720
10      50000      Spandau      25564      247
      144 ...      111      2379
      226      645      6473
13 11      60000      Steglitz-Zehlendorf      23719      213
      129 ...      52      2698
      481      397      5422
14
15 [12 rows x 19 columns]
16 Command being timed: "python testcase1/prompt2/exec3/script.py"
17 User time (seconds): 2.36
18 System time (seconds): 0.10
19 Percent of CPU this job got: 498%
20 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.49
21 Average shared text size (kbytes): 0
22 Average unshared data size (kbytes): 0
23 Average stack size (kbytes): 0
24 Average total size (kbytes): 0
25 Maximum resident set size (kbytes): 153500
26 Average resident set size (kbytes): 0
27 Major (requiring I/O) page faults: 0
28 Minor (reclaiming a frame) page faults: 32482
29 Voluntary context switches: 36
30 Involuntary context switches: 215
31 Swaps: 0
32 File system inputs: 0
33 File system outputs: 0
34 Socket messages sent: 0
35 Socket messages received: 0
36 Signals delivered: 0
37 Page size (bytes): 4096
38 Exit status: 0

```

Listing 200: Ausgabe Testfall 1 Prompt 2 Ausführung 3

```

1 Ursprüngliche Daten:
2   LOR-Schlüssel      Bezirke      Straftaten_insgesamt
   Raub ...      Sachbeschädigung_insgesamt      Sachbeschädigung_Graffiti
   Rauschgiftdelikte      Kieztaten
3 0      10000      Mitte      83611
   884 ...      6172      976
   3306      19212
4 1      20000      Friedrichshain-Kreuzberg      55838
   923 ...      4906      801
   4455      12724
5 2      30000      Pankow      39164
   529 ...      4576      942
   910      10043
6 3      40000      Charlottenburg-Wilmersdorf      53630
   464 ...      4134      750
   1431      12175
7 4      50000      Spandau      25564
   247 ...      2379      226

```

		645	6473			
8	5	60000		Steglitz-Zehlendorf		23719
		213 ...		2698	481	
		397	5422			
9	6	70000		Tempelhof-Schöneberg		41776
		328 ...		3592	622	
		1435	10132			
10	7	80000		Neukölln		45313
		508 ...		4165	379	
		1869	10253			
11	8	90000		Treptow-Köpenick		26099
		207 ...		3152	558	
		680	6720			
12	9	100000		Marzahn-Hellersdorf		26649
		315 ...		2908	392	
		424	8203			
13	10	110000		Lichtenberg		31563
		362 ...		3206	628	
		663	8499			
14	11	120000		Reinickendorf		27344
		311 ...		2481	203	
		553	7323			
15	12	999900	Stadtgebiet Berlin, nicht zuzuordnen			56425
		98 ...		1065	490	
		734	2558			
16	13	999999		Berlin (PKS gesamt)		536695
		5389 ...		45434	7448	
		17502	119737			
17						
18		[14 rows x 19 columns]				
19						
20		Gefilterte Daten (ohne nicht zuzuordnende Bezirke):				
21		LOR-Schlüssel		Bezirke	Straftaten_insgesamt	Raub
		Straßenraub ...	Brandstiftung	Sachbeschädigung_insgesamt	Sachbeschä	
		digung_Graffiti	Rauschgiftdelikte	Kieztaten		
22	0	10000		Mitte	83611	884
		521 ...	137		6172	
		976	3306	19212		
23	1	20000	Friedrichshain-Kreuzberg		55838	923
		606 ...	85		4906	
		801	4455	12724		
24	2	30000		Pankow	39164	529
		389 ...	99		4576	
		942	910	10043		
25	3	40000	Charlottenburg-Wilmersdorf		53630	464
		278 ...	79		4134	
		750	1431	12175		
26	4	50000		Spandau	25564	247
		144 ...	111		2379	
		226	645	6473		
27	5	60000	Steglitz-Zehlendorf		23719	213
		129 ...	52		2698	

		481	397	5422		
28	6	70000	Tempelhof-Schöneberg		41776	328
		198 ...	86		3592	
		622	1435	10132		
29	7	80000	Neukölln		45313	508
		294 ...	145		4165	
		379	1869	10253		
30	8	90000	Treptow-Köpenick		26099	207
		118 ...	77		3152	
		558	680	6720		
31	9	100000	Marzahn-Hellersdorf		26649	315
		177 ...	96		2908	
		392	424	8203		
32	10	110000	Lichtenberg		31563	362
		259 ...	79		3206	
		628	663	8499		
33	11	120000	Reinickendorf		27344	311
		173 ...	76		2481	
		203	553	7323		
34						
35		[12 rows x 19 columns]				
36						
37		Sortierte Daten nach 'Straftaten_insgesamt':				
38		LOR-Schlüssel	Bezirke	Straftaten_insgesamt	Raub	
		Straßenraub ...	Brandstiftung	Sachbeschädigung_insgesamt	Sachbeschä	
		digung_Graffiti	Rauschgiftdelikte	Kieztaten		
39	0	10000	Mitte		83611	884
		521 ...	137		6172	
		976	3306	19212		
40	1	20000	Friedrichshain-Kreuzberg		55838	923
		606 ...	85		4906	
		801	4455	12724		
41	2	40000	Charlottenburg-Wilmersdorf		53630	464
		278 ...	79		4134	
		750	1431	12175		
42	3	80000	Neukölln		45313	508
		294 ...	145		4165	
		379	1869	10253		
43	4	70000	Tempelhof-Schöneberg		41776	328
		198 ...	86		3592	
		622	1435	10132		
44	5	30000	Pankow		39164	529
		389 ...	99		4576	
		942	910	10043		
45	6	110000	Lichtenberg		31563	362
		259 ...	79		3206	
		628	663	8499		
46	7	120000	Reinickendorf		27344	311
		173 ...	76		2481	
		203	553	7323		
47	8	100000	Marzahn-Hellersdorf		26649	315
		177 ...	96		2908	

```

48 9          392          424      8203
          90000          Treptow-Köpenick          26099      207
          118 ...          77          3152
          558          680      6720
49 10          50000          Spandau          25564      247
          144 ...          111          2379
          226          645      6473
50 11          60000          Steglitz-Zehlendorf          23719      213
          129 ...          52          2698
          481          397      5422
51
52 [12 rows x 19 columns]
53 Command being timed: "python testcase1/prompt2/exec4/script.py"
54 User time (seconds): 2.40
55 System time (seconds): 0.06
56 Percent of CPU this job got: 483%
57 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.51
58 Average shared text size (kbytes): 0
59 Average unshared data size (kbytes): 0
60 Average stack size (kbytes): 0
61 Average total size (kbytes): 0
62 Maximum resident set size (kbytes): 153192
63 Average resident set size (kbytes): 0
64 Major (requiring I/O) page faults: 0
65 Minor (reclaiming a frame) page faults: 32485
66 Voluntary context switches: 40
67 Involuntary context switches: 429
68 Swaps: 0
69 File system inputs: 64
70 File system outputs: 0
71 Socket messages sent: 0
72 Socket messages received: 0
73 Signals delivered: 0
74 Page size (bytes): 4096
75 Exit status: 0

```

Listing 201: Ausgabe Testfall 1Prompt 2Ausführung 4

```

1 Excel-Datei erfolgreich eingelesen.
2
3 Erste 5 Zeilen der Originaldaten:
4   LOR-Schlüssel          Bezirke  Straftaten_insgesamt  Raub
   Straßenraub ...  Brandstiftung  Sachbeschädigung_insgesamt  Sachbeschä
   digung_Graffiti  Rauschgiftdelikte  Kieztaten
5 0          10000          Mitte          83611      884
          521 ...          137          6172
          976          3306      19212
6 1          20000  Friedrichshain-Kreuzberg          55838      923
          606 ...          85          4906
          801          4455      12724
7 2          30000          Pankow          39164      529
          389 ...          99          4576
          942          910      10043

```

```

8 3      40000  Charlottenburg-Wilmersdorf      53630  464
      278 ...      79      4134
      750      1431      12175
9 4      50000      Spandau      25564  247
      144 ...      111      2379
      226      645      6473
10
11 [5 rows x 19 columns]
12
13 Anzahl der Zeilen nach dem Ausschluss: 12
14
15 Erste 5 Zeilen der sortierten Daten:
16 LOR-Schlüssel      Bezirke  Straftaten_insgesamt  Raub
Straßenraub ... Brandstiftung Sachbeschädigung_insgesamt Sachbeschä
digung_Graffiti Rauschgiftdelikte Kieztaten
17 0      10000      Mitte      83611  884
      521 ...      137      6172
      976      3306      19212
18 1      20000  Friedrichshain-Kreuzberg      55838  923
      606 ...      85      4906
      801      4455      12724
19 2      40000  Charlottenburg-Wilmersdorf      53630  464
      278 ...      79      4134
      750      1431      12175
20 3      80000      Neukölln      45313  508
      294 ...      145      4165
      379      1869      10253
21 4      70000  Tempelhof-Schöneberg      41776  328
      198 ...      86      3592
      622      1435      10132
22
23 [5 rows x 19 columns]
24
25 Sortierte Daten wurden erfolgreich in 'Fallzahlen_sortiert.xlsx' gespeichert.
26 Command being timed: "python testcase1/prompt2/exec5/script.py"
27 User time (seconds): 2.40
28 System time (seconds): 0.06
29 Percent of CPU this job got: 492%
30 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.50
31 Average shared text size (kbytes): 0
32 Average unshared data size (kbytes): 0
33 Average stack size (kbytes): 0
34 Average total size (kbytes): 0
35 Maximum resident set size (kbytes): 153760
36 Average resident set size (kbytes): 0
37 Major (requiring I/O) page faults: 0
38 Minor (reclaiming a frame) page faults: 32531
39 Voluntary context switches: 37
40 Involuntary context switches: 350
41 Swaps: 0
42 File system inputs: 0
43 File system outputs: 112

```

```
44 Socket messages sent: 0
45 Socket messages received: 0
46 Signals delivered: 0
47 Page size (bytes): 4096
48 Exit status: 0
```

Listing 202: Ausgabe Testfall 1Prompt 2Ausführung 5

```
1 Erfolgreich das Blatt 'Fallzahlen_2023' aus 'Fallzahlen.xlsx' gelesen.
2 Alle erforderlichen Spalten sind vorhanden.
3 DataFrame erfolgreich sortiert.
4 Das sortierte DataFrame wurde erfolgreich in 'Fallzahlen_sortiert.xlsx'
  gespeichert.
5 Command being timed: "python testcase1/prompt3/exec1/script.py"
6 User time (seconds): 2.44
7 System time (seconds): 0.06
8 Percent of CPU this job got: 456%
9 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.55
10 Average shared text size (kbytes): 0
11 Average unshared data size (kbytes): 0
12 Average stack size (kbytes): 0
13 Average total size (kbytes): 0
14 Maximum resident set size (kbytes): 153732
15 Average resident set size (kbytes): 0
16 Major (requiring I/O) page faults: 0
17 Minor (reclaiming a frame) page faults: 32469
18 Voluntary context switches: 42
19 Involuntary context switches: 309
20 Swaps: 0
21 File system inputs: 72
22 File system outputs: 80
23 Socket messages sent: 0
24 Socket messages received: 0
25 Signals delivered: 0
26 Page size (bytes): 4096
27 Exit status: 0
```

Listing 203: Ausgabe Testfall 1Prompt 3Ausführung 1

```
1 Excel-Datei 'Fallzahlen.xlsx' erfolgreich eingelesen.
2 Daten erfolgreich sortiert.
3 Sortiertes DataFrame wurde erfolgreich in 'Sortierte_Fallzahlen.xlsx'
  gespeichert.
4 Command being timed: "python testcase1/prompt3/exec2/script.py"
5 User time (seconds): 2.46
6 System time (seconds): 0.05
7 Percent of CPU this job got: 458%
8 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.54
9 Average shared text size (kbytes): 0
10 Average unshared data size (kbytes): 0
11 Average stack size (kbytes): 0
12 Average total size (kbytes): 0
13 Maximum resident set size (kbytes): 153792
```



```

14 Average resident set size (kbytes): 0
15 Major (requiring I/O) page faults: 0
16 Minor (reclaiming a frame) page faults: 32508
17 Voluntary context switches: 36
18 Involuntary context switches: 269
19 Swaps: 0
20 File system inputs: 0
21 File system outputs: 80
22 Socket messages sent: 0
23 Socket messages received: 0
24 Signals delivered: 0
25 Page size (bytes): 4096
26 Exit status: 0

```

Listing 204: Ausgabe Testfall 1Prompt 3Ausführung 2

```

1 Excel-Datei erfolgreich eingelesen.
2 Sortiertes DataFrame:
3   LOR-Schlüssel      Bezirke  Straftaten_insgesamt  Raub
   Straßenraub ... Brandstiftung  Sachbeschädigung_insgesamt  Sachbeschä
   digung_Graffiti  Rauschgiftdelikte  Kieztaten
4 0      10000      Mitte      83611      884
   521 ...      137      6172
   976      3306      19212
5 1      20000  Friedrichshain-Kreuzberg      55838      923
   606 ...      85      4906
   801      4455      12724
6 2      40000  Charlottenburg-Wilmersdorf      53630      464
   278 ...      79      4134
   750      1431      12175
7 3      80000      Neukölln      45313      508
   294 ...      145      4165
   379      1869      10253
8 4      70000  Tempelhof-Schöneberg      41776      328
   198 ...      86      3592
   622      1435      10132
9
10 [5 rows x 19 columns]
11 Das sortierte DataFrame wurde erfolgreich in 'Fallzahlen_sortiert.xlsx'
   gespeichert.
12 Command being timed: "python testcase1/prompt3/exec3/script.py"
13 User time (seconds): 2.44
14 System time (seconds): 0.06
15 Percent of CPU this job got: 448%
16 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.56
17 Average shared text size (kbytes): 0
18 Average unshared data size (kbytes): 0
19 Average stack size (kbytes): 0
20 Average total size (kbytes): 0
21 Maximum resident set size (kbytes): 153764
22 Average resident set size (kbytes): 0
23 Major (requiring I/O) page faults: 0
24 Minor (reclaiming a frame) page faults: 32481

```

```

25 Voluntary context switches: 34
26 Involuntary context switches: 158
27 Swaps: 0
28 File system inputs: 0
29 File system outputs: 176
30 Socket messages sent: 0
31 Socket messages received: 0
32 Signals delivered: 0
33 Page size (bytes): 4096
34 Exit status: 0

```

Listing 205: Ausgabe Testfall 1Prompt 3Ausführung 3

```

1 Successfully read the sheet 'Fallzahlen_2023' from 'Fallzahlen.xlsx'.
2 Successfully sorted the included data by 'Straftaten_insgesamt' in descending
  order.
3 Successfully concatenated the excluded rows at the end of the sorted data.
4 Sorted DataFrame:
5      LOR-Schlüssel      Bezirke  Straftaten_insgesamt
   Raub ... Sachbeschädigung_insgesamt  Sachbeschädigung_Graffiti
   Rauschgiftdelikte  Kieztaten
6 0      10000      Mitte      83611
   884 ...      6172      976
   3306      19212
7 1      20000      Friedrichshain-Kreuzberg      55838
   923 ...      4906      801
   4455      12724
8 2      40000      Charlottenburg-Wilmersdorf      53630
   464 ...      4134      750
   1431      12175
9 3      80000      Neukölln      45313
   508 ...      4165      379
   1869      10253
10 4      70000      Tempelhof-Schöneberg      41776
   328 ...      3592      622
   1435      10132
11 5      30000      Pankow      39164
   529 ...      4576      942
   910      10043
12 6      110000      Lichtenberg      31563
   362 ...      3206      628
   663      8499
13 7      120000      Reinickendorf      27344
   311 ...      2481      203
   553      7323
14 8      100000      Marzahn-Hellersdorf      26649
   315 ...      2908      392
   424      8203
15 9      90000      Treptow-Köpenick      26099
   207 ...      3152      558
   680      6720
16 10      50000      Spandau      25564
   247 ...      2379      226

```

```

17 11      645      6473
      60000      Steglitz-Zehlendorf      23719
      213 ...      2698      481
      397      5422
18 12      999900 Stadtgebiet Berlin, nicht zuzuordnen      56425
      98 ...      1065      490
      734      2558
19 13      999999      Berlin (PKS gesamt)      536695
      5389 ...      45434      7448
      17502      119737
20
21 [14 rows x 19 columns]
22 Command being timed: "python testcase1/prompt3/exec4/script.py"
23 User time (seconds): 2.46
24 System time (seconds): 0.05
25 Percent of CPU this job got: 445%
26 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.56
27 Average shared text size (kbytes): 0
28 Average unshared data size (kbytes): 0
29 Average stack size (kbytes): 0
30 Average total size (kbytes): 0
31 Maximum resident set size (kbytes): 153708
32 Average resident set size (kbytes): 0
33 Major (requiring I/O) page faults: 0
34 Minor (reclaiming a frame) page faults: 32473
35 Voluntary context switches: 41
36 Involuntary context switches: 257
37 Swaps: 0
38 File system inputs: 0
39 File system outputs: 0
40 Socket messages sent: 0
41 Socket messages received: 0
42 Signals delivered: 0
43 Page size (bytes): 4096
44 Exit status: 0

```

Listing 206: Ausgabe Testfall 1Prompt 3Ausführung 4

```

1 Erfolgreich das Sheet 'Fallzahlen_2023' aus 'Fallzahlen.xlsx' eingelesen.
2 Alle erforderlichen Spalten sind vorhanden.
3 Anzahl der Bezirke: 12
4 Anzahl der nicht zuzuordnenden Einträge: 2
5 Bezirke erfolgreich nach 'Straftaten_insgesamt' absteigend sortiert.
6 Bezirke und nicht zuzuordnende Einträge erfolgreich kombiniert.
7 Daten erfolgreich sortiert.
8 LOR-Schlüssel      Bezirke Straftaten_insgesamt Raub
  Straßenraub ... Brandstiftung Sachbeschädigung_insgesamt Sachbeschä
  digung_Graffiti Rauschgiftdelikte Kieztaten
9 0      10000      Mitte      83611      884
      521 ...      137      6172
      976      3306      19212
10 1      20000 Friedrichshain-Kreuzberg      55838      923
      606 ...      85      4906

```

```

11 2      801      4455      12724
      40000 Charlottenburg-Wilmersdorf      53630      464
      278 ...      79      4134
      750      1431      12175
12 3      80000      Neukölln      45313      508
      294 ...      145      4165
      379      1869      10253
13 4      70000 Tempelhof-Schöneberg      41776      328
      198 ...      86      3592
      622      1435      10132
14
15 [5 rows x 19 columns]
16 Command being timed: "python testcase1/prompt3/exec5/script.py"
17 User time (seconds): 2.39
18 System time (seconds): 0.05
19 Percent of CPU this job got: 463%
20 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.52
21 Average shared text size (kbytes): 0
22 Average unshared data size (kbytes): 0
23 Average stack size (kbytes): 0
24 Average total size (kbytes): 0
25 Maximum resident set size (kbytes): 153460
26 Average resident set size (kbytes): 0
27 Major (requiring I/O) page faults: 0
28 Minor (reclaiming a frame) page faults: 32514
29 Voluntary context switches: 36
30 Involuntary context switches: 205
31 Swaps: 0
32 File system inputs: 0
33 File system outputs: 0
34 Socket messages sent: 0
35 Socket messages received: 0
36 Signals delivered: 0
37 Page size (bytes): 4096
38 Exit status: 0

```

Listing 207: Ausgabe Testfall 1Prompt 3Ausführung 5

11.3.15 Komplette Ausgaben der Skripte Testfall 2

```

1      LOR-Schlüssel      Bezirke      Straftaten_insgesamt      Raub      Straß
enraub ... Brandstiftung Sachbeschädigung_insgesamt Sachbeschä
digung_Graffiti Rauschgiftdelikte Kieztaten
2 0      999999 Berlin (PKS gesamt)      569549      5407
      2951 ...      841      41954
      9894      15753      102589
3 1      999999 Berlin (PKS gesamt)      568860      5156
      3015 ...      1039      44066
      10501      14880      102793
4 2      999999 Berlin (PKS gesamt)      543156      5697
      3287 ...      917      42894
      9239      13465      100363

```

```

5 3          999999 Berlin (PKS gesamt)          536695 5389
   3333 ...          1125          45434
   7448          17502          119737
6 4          999999 Berlin (PKS gesamt)          520437 4242
   2354 ...          874          41804
   10913          16077          96416
7 ..          ...          ...          ...
   ... ...          ...          ...
   ...          ...          ...
8 135          90000 Treptow-Köpenick          23250 209
   132 ...          72          2517
   576          392          4729
9 136          100000 Marzahn-Hellersdorf          23245 222
   121 ...          44          2655
   505          367          5720
10 137          60000 Steglitz-Zehlendorf          22879 178
   116 ...          61          2969
   877          495          5417
11 138          100000 Marzahn-Hellersdorf          22830 289
   174 ...          81          2791
   540          588          6089
12 139          100000 Marzahn-Hellersdorf          22413 233
   169 ...          52          2865
   470          488          6035
13
14 [140 rows x 19 columns]
15 Command being timed: "python testcase2/prompt1/exec1/script.py"
16 User time (seconds): 4.66
17 System time (seconds): 0.25
18 Percent of CPU this job got: 165%
19 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:02.96
20 Average shared text size (kbytes): 0
21 Average unshared data size (kbytes): 0
22 Average stack size (kbytes): 0
23 Average total size (kbytes): 0
24 Maximum resident set size (kbytes): 154176
25 Average resident set size (kbytes): 0
26 Major (requiring I/O) page faults: 0
27 Minor (reclaiming a frame) page faults: 167789
28 Voluntary context switches: 35
29 Involuntary context switches: 377
30 Swaps: 0
31 File system inputs: 0
32 File system outputs: 0
33 Socket messages sent: 0
34 Socket messages received: 0
35 Signals delivered: 0
36 Page size (bytes): 4096
37 Exit status: 0

```

Listing 208: Ausgabe Testfall 2Prompt 1Ausführung 1

1	Bezirke	Straftaten_insgesamt
---	---------	----------------------

```

2 0          Berlin (PKS gesamt)          5269896
3 1              Mitte                    878048
4 2          Friedrichshain-Kreuzberg     579591
5 3          Charlottenburg-Wilmersdorf   519706
6 4 Stadtgebiet Berlin, nicht zuzuordnen  512158
7 5              Neukölln                 436568
8 6          Tempelhof-Schöneberg         398767
9 7              Pankow                   393574
10 8          Reinickendorf               284193
11 9          Lichtenberg                 279018
12 10             Spandau                 257757
13 11          Treptow-Köpenick           247383
14 12          Marzahn-Hellersdorf         241928
15 13          Steglitz-Zehlendorf        241205
16 Command being timed: "python testcase2/prompt1/exec2/script.py"
17 User time (seconds): 4.81
18 System time (seconds): 0.25
19 Percent of CPU this job got: 161%
20 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:03.13
21 Average shared text size (kbytes): 0
22 Average unshared data size (kbytes): 0
23 Average stack size (kbytes): 0
24 Average total size (kbytes): 0
25 Maximum resident set size (kbytes): 153660
26 Average resident set size (kbytes): 0
27 Major (requiring I/O) page faults: 0
28 Minor (reclaiming a frame) page faults: 168290
29 Voluntary context switches: 34
30 Involuntary context switches: 293
31 Swaps: 0
32 File system inputs: 0
33 File system outputs: 0
34 Socket messages sent: 0
35 Socket messages received: 0
36 Signals delivered: 0
37 Page size (bytes): 4096
38 Exit status: 0

```

Listing 209: Ausgabe Testfall 2Prompt 1Ausführung 2

```

1          Bezirke  Straftaten_insgesamt
2 0          Berlin (PKS gesamt)          5269896
3 1              Mitte                    878048
4 2          Friedrichshain-Kreuzberg     579591
5 3          Charlottenburg-Wilmersdorf   519706
6 4 Stadtgebiet Berlin, nicht zuzuordnen  512158
7 5              Neukölln                 436568
8 6          Tempelhof-Schöneberg         398767
9 7              Pankow                   393574
10 8          Reinickendorf               284193
11 9          Lichtenberg                 279018
12 10             Spandau                 257757
13 11          Treptow-Köpenick           247383

```

```

14 12 Marzahn-Hellersdorf 241928
15 13 Steglitz-Zehlendorf 241205
16 Command being timed: "python testcase2/prompt1/exec3/script.py"
17 User time (seconds): 4.79
18 System time (seconds): 0.22
19 Percent of CPU this job got: 160%
20 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:03.13
21 Average shared text size (kbytes): 0
22 Average unshared data size (kbytes): 0
23 Average stack size (kbytes): 0
24 Average total size (kbytes): 0
25 Maximum resident set size (kbytes): 154584
26 Average resident set size (kbytes): 0
27 Major (requiring I/O) page faults: 0
28 Minor (reclaiming a frame) page faults: 168796
29 Voluntary context switches: 35
30 Involuntary context switches: 281
31 Swaps: 0
32 File system inputs: 0
33 File system outputs: 0
34 Socket messages sent: 0
35 Socket messages received: 0
36 Signals delivered: 0
37 Page size (bytes): 4096
38 Exit status: 0

```

Listing 210: Ausgabe Testfall 2Prompt 1Ausführung 3

```

1      Bezirke  Straftaten_insgesamt
2 0      Berlin (PKS gesamt) 5269896
3 1      Mitte 878048
4 2      Friedrichshain-Kreuzberg 579591
5 3      Charlottenburg-Wilmersdorf 519706
6 4      Stadtgebiet Berlin, nicht zuzuordnen 512158
7 5      Neukölln 436568
8 6      Tempelhof-Schöneberg 398767
9 7      Pankow 393574
10 8      Reinickendorf 284193
11 9      Lichtenberg 279018
12 10      Spandau 257757
13 11      Treptow-Köpenick 247383
14 12      Marzahn-Hellersdorf 241928
15 13      Steglitz-Zehlendorf 241205
16 Command being timed: "python testcase2/prompt1/exec4/script.py"
17 User time (seconds): 4.59
18 System time (seconds): 0.26
19 Percent of CPU this job got: 167%
20 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:02.90
21 Average shared text size (kbytes): 0
22 Average unshared data size (kbytes): 0
23 Average stack size (kbytes): 0
24 Average total size (kbytes): 0
25 Maximum resident set size (kbytes): 153620

```

```

26 Average resident set size (kbytes): 0
27 Major (requiring I/O) page faults: 0
28 Minor (reclaiming a frame) page faults: 168213
29 Voluntary context switches: 47
30 Involuntary context switches: 429
31 Swaps: 0
32 File system inputs: 0
33 File system outputs: 176
34 Socket messages sent: 0
35 Socket messages received: 0
36 Signals delivered: 0
37 Page size (bytes): 4096
38 Exit status: 0

```

Listing 211: Ausgabe Testfall 2 Prompt 1 Ausführung 4

```

1 Anzahl der gelesenen Sheets: 10
2 Anzahl der Zeilen nach dem Kombinieren: 140
3 Die Daten wurden erfolgreich sortiert.
4   LOR-Schlüssel           Bezirke   Straftaten_insgesamt   Raub   Straß
   enraub ... Brandstiftung Sachbeschädigung_insgesamt Sachbeschä
   digung_Graffiti Rauschgiftdelikte Kieztaten
5 0      999999 Berlin (PKS gesamt)           569549  5407
   2951 ...           841           41954
   9894           15753   102589
6 1      999999 Berlin (PKS gesamt)           568860  5156
   3015 ...           1039           44066
   10501           14880   102793
7 2      999999 Berlin (PKS gesamt)           543156  5697
   3287 ...           917           42894
   9239           13465   100363
8 3      999999 Berlin (PKS gesamt)           536695  5389
   3333 ...           1125           45434
   7448           17502   119737
9 4      999999 Berlin (PKS gesamt)           520437  4242
   2354 ...           874           41804
   10913           16077   96416
10 ..      ...           ...           ...
   ... ...           ...           ...
   ...           ...           ...
11 135      90000 Treptow-Köpenick           23250   209
   132 ...           72           2517
   576           392   4729
12 136      100000 Marzahn-Hellersdorf           23245   222
   121 ...           44           2655
   505           367   5720
13 137      60000 Steglitz-Zehlendorf           22879   178
   116 ...           61           2969
   877           495   5417
14 138      100000 Marzahn-Hellersdorf           22830   289
   174 ...           81           2791
   540           588   6089
15 139      100000 Marzahn-Hellersdorf           22413   233

```



```

16      169    ...           52           2865
17      470           488       6035
18 [140 rows x 19 columns]
19 Command being timed: "python testcase2/prompt1/exec5/script.py"
20 User time (seconds): 4.52
21 System time (seconds): 0.24
22 Percent of CPU this job got: 167%
23 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:02.85
24 Average shared text size (kbytes): 0
25 Average unshared data size (kbytes): 0
26 Average stack size (kbytes): 0
27 Average total size (kbytes): 0
28 Maximum resident set size (kbytes): 153984
29 Average resident set size (kbytes): 0
30 Major (requiring I/O) page faults: 0
31 Minor (reclaiming a frame) page faults: 167837
32 Voluntary context switches: 34
33 Involuntary context switches: 329
34 Swaps: 0
35 File system inputs: 0
36 File system outputs: 0
37 Socket messages sent: 0
38 Socket messages received: 0
39 Signals delivered: 0
40 Page size (bytes): 4096
Exit status: 0

```

Listing 212: Ausgabe Testfall 2Prompt 1Ausführung 5

```

1 Gesheet 'Fallzahlen_2014' erfolgreich geladen.
2 Gesheet 'Fallzahlen_2015' erfolgreich geladen.
3 Gesheet 'Fallzahlen_2016' erfolgreich geladen.
4 Gesheet 'Fallzahlen_2017' erfolgreich geladen.
5 Gesheet 'Fallzahlen_2018' erfolgreich geladen.
6 Gesheet 'Fallzahlen_2019' erfolgreich geladen.
7 Gesheet 'Fallzahlen_2020' erfolgreich geladen.
8 Gesheet 'Fallzahlen_2021' erfolgreich geladen.
9 Gesheet 'Fallzahlen_2022' erfolgreich geladen.
10 Gesheet 'Fallzahlen_2023' erfolgreich geladen.
11 Alle Sheets wurden erfolgreich zusammengeführt.
12 Unerwünschte LOR-Schlüssel wurden entfernt.
13 DataFrame wurde nach 'Straftaten_insgesamt' sortiert.
14 LOR-Schlüssel Bezirke Straftaten_insgesamt Raub Straßenraub ...
   Brandstiftung Sachbeschädigung_insgesamt Sachbeschädigung_Graffiti
   Rauschgiftdelikte Kieztaten
15 0      10000  Mitte      106006  976      611 ...
      112      5598      1330
      3350      16015
16 1      10000  Mitte      98046  1040      589 ...
      110      5256      1245
      2638      15078
17 2      10000  Mitte      96716  763      465 ...

```

```

18 3      113      5625      1488
      3839      15900
      10000      Mitte      91321      1014      521 ...
      125      5406      1177
      2880      15703
19 4      10000      Mitte      85227      733      449 ...
      75      5528      1510
      3792      14899
20
21 [5 rows x 19 columns]
22 Command being timed: "python testcase2/prompt2/exec1/script.py"
23 User time (seconds): 4.68
24 System time (seconds): 0.24
25 Percent of CPU this job got: 162%
26 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:03.03
27 Average shared text size (kbytes): 0
28 Average unshared data size (kbytes): 0
29 Average stack size (kbytes): 0
30 Average total size (kbytes): 0
31 Maximum resident set size (kbytes): 155928
32 Average resident set size (kbytes): 0
33 Major (requiring I/O) page faults: 0
34 Minor (reclaiming a frame) page faults: 168420
35 Voluntary context switches: 38
36 Involuntary context switches: 687
37 Swaps: 0
38 File system inputs: 0
39 File system outputs: 0
40 Socket messages sent: 0
41 Socket messages received: 0
42 Signals delivered: 0
43 Page size (bytes): 4096
44 Exit status: 0

```

Listing 213: Ausgabe Testfall 2Prompt 2Ausführung 1

```

1 Lade Sheet: Fallzahlen_2014 mit 14 Zeilen.
2 Lade Sheet: Fallzahlen_2015 mit 14 Zeilen.
3 Lade Sheet: Fallzahlen_2016 mit 14 Zeilen.
4 Lade Sheet: Fallzahlen_2017 mit 14 Zeilen.
5 Lade Sheet: Fallzahlen_2018 mit 14 Zeilen.
6 Lade Sheet: Fallzahlen_2019 mit 14 Zeilen.
7 Lade Sheet: Fallzahlen_2020 mit 14 Zeilen.
8 Lade Sheet: Fallzahlen_2021 mit 14 Zeilen.
9 Lade Sheet: Fallzahlen_2022 mit 14 Zeilen.
10 Lade Sheet: Fallzahlen_2023 mit 14 Zeilen.
11 Gesamtanzahl der Zeilen nach dem Zusammenführen: 140
12 Anzahl der Zeilen nach dem Filtern: 140
13 Daten wurden nach 'Straftaten_insgesamt' sortiert.
14
15 Ergebnis:
16 LOR-Schlüssel      Bezirke  Straftaten_insgesamt  Raub  Straßenraub
... Brandstiftung  Sachbeschädigung_insgesamt  Sachbeschä

```

```

17 0      digung_Graffiti  Rauschgiftdelikte  Kieztaten
      999999  Berlin (PKS gesamt)      569549  5407      2951
      ...      841      41954
      9894      15753      102589
18 1      999999  Berlin (PKS gesamt)      568860  5156      3015
      ...      1039      44066
      10501      14880      102793
19 2      999999  Berlin (PKS gesamt)      543156  5697      3287
      ...      917      42894
      9239      13465      100363
20 3      999999  Berlin (PKS gesamt)      536695  5389      3333
      ...      1125      45434
      7448      17502      119737
21 4      999999  Berlin (PKS gesamt)      520437  4242      2354
      ...      874      41804
      10913      16077      96416
22
23 [5 rows x 19 columns]
24 Command being timed: "python testcase2/prompt2/exec2/script.py"
25 User time (seconds): 4.64
26 System time (seconds): 0.25
27 Percent of CPU this job got: 165%
28 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:02.96
29 Average shared text size (kbytes): 0
30 Average unshared data size (kbytes): 0
31 Average stack size (kbytes): 0
32 Average total size (kbytes): 0
33 Maximum resident set size (kbytes): 154160
34 Average resident set size (kbytes): 0
35 Major (requiring I/O) page faults: 0
36 Minor (reclaiming a frame) page faults: 167518
37 Voluntary context switches: 33
38 Involuntary context switches: 376
39 Swaps: 0
40 File system inputs: 0
41 File system outputs: 0
42 Socket messages sent: 0
43 Socket messages received: 0
44 Signals delivered: 0
45 Page size (bytes): 4096
46 Exit status: 0

```

Listing 214: Ausgabe Testfall 2Prompt 2Ausführung 2

```

1      LOR-Schlüssel      Bezirke  Straftaten_insgesamt  Raub
      ...  Sachbeschädigung_Graffiti_dup  Rauschgiftdelikte_dup  Kieztaten_dup
      Sheet_dup
2 0      10000      Mitte      91321  1014
      ...      1245      2638      15078
      Fallzahlen_2015
3 1      20000  Friedrichshain-Kreuzberg      62194  940
      ...      1338      5571      11223
      Fallzahlen_2015

```

4	2	40000	Charlottenburg-Wilmersdorf	53322	641
	...		653	885	10407
		Fallzahlen_2015			
5	3	80000	Neukölln	44978	580
	...		670	1683	9854
		Fallzahlen_2015			
6	4	70000	Tempelhof-Schöneberg	40184	496
	...		702	900	7933
		Fallzahlen_2015			
7	5	30000	Pankow	39803	313
	...		1312	510	11470
		Fallzahlen_2015			
8	6	120000	Reinickendorf	29180	340
	...		649	869	6959
		Fallzahlen_2015			
9	7	50000	Spandau	26499	344
	...		357	493	6631
		Fallzahlen_2015			
10	8	110000	Lichtenberg	25618	236
	...		568	387	5669
		Fallzahlen_2015			
11	9	60000	Steglitz-Zehlendorf	25072	210
	...		651	356	4478
		Fallzahlen_2015			
12	10	100000	Marzahn-Hellersdorf	24285	260
	...		505	367	5720
		Fallzahlen_2015			
13	11	90000	Treptow-Köpenick	23250	209
	...		595	517	4798
		Fallzahlen_2015			

14
15 [12 rows x 38 columns]

16 Command being timed: "python testcase2/prompt2/exec3/script.py"

17 User time (seconds): 4.69

18 System time (seconds): 0.24

19 Percent of CPU this job got: 165%

20 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:02.99

21 Average shared text size (kbytes): 0

22 Average unshared data size (kbytes): 0

23 Average stack size (kbytes): 0

24 Average total size (kbytes): 0

25 Maximum resident set size (kbytes): 153804

26 Average resident set size (kbytes): 0

27 Major (requiring I/O) page faults: 0

28 Minor (reclaiming a frame) page faults: 168209

29 Voluntary context switches: 50

30 Involuntary context switches: 415

31 Swaps: 0

32 File system inputs: 0

33 File system outputs: 176

34 Socket messages sent: 0

35 Socket messages received: 0

```
36 Signals delivered: 0
37 Page size (bytes): 4096
38 Exit status: 0
```

Listing 215: Ausgabe Testfall 2Prompt 2Ausführung 3

```
1 Es wurden 10 Sheets gefunden und geladen.
2 Verarbeite Sheet: Fallzahlen_2014
3 Anzahl der Bezirke nach Filtern in Sheet 'Fallzahlen_2014': 12
4 Verarbeite Sheet: Fallzahlen_2015
5 Anzahl der Bezirke nach Filtern in Sheet 'Fallzahlen_2015': 12
6 Verarbeite Sheet: Fallzahlen_2016
7 Anzahl der Bezirke nach Filtern in Sheet 'Fallzahlen_2016': 12
8 Verarbeite Sheet: Fallzahlen_2017
9 Anzahl der Bezirke nach Filtern in Sheet 'Fallzahlen_2017': 12
10 Verarbeite Sheet: Fallzahlen_2018
11 Anzahl der Bezirke nach Filtern in Sheet 'Fallzahlen_2018': 12
12 Verarbeite Sheet: Fallzahlen_2019
13 Anzahl der Bezirke nach Filtern in Sheet 'Fallzahlen_2019': 12
14 Verarbeite Sheet: Fallzahlen_2020
15 Anzahl der Bezirke nach Filtern in Sheet 'Fallzahlen_2020': 12
16 Verarbeite Sheet: Fallzahlen_2021
17 Anzahl der Bezirke nach Filtern in Sheet 'Fallzahlen_2021': 12
18 Verarbeite Sheet: Fallzahlen_2022
19 Anzahl der Bezirke nach Filtern in Sheet 'Fallzahlen_2022': 12
20 Verarbeite Sheet: Fallzahlen_2023
21 Anzahl der Bezirke nach Filtern in Sheet 'Fallzahlen_2023': 12
22 Gesamtanzahl der Zeilen nach dem Zusammenfügen: 120
23 Daten wurden nach 'LOR-Schlüssel' und 'Bezirke' aggregiert.
24 Daten wurden nach 'Straftaten_insgesamt' sortiert.
25
26 Sortiertes Ergebnis:
27   LOR-Schlüssel           Bezirke  Straftaten_insgesamt
28 0           10000           Mitte           878048
29 1           20000 Friedrichshain-Kreuzberg       579591
30 2           40000 Charlottenburg-Wilmersdorf     519706
31 3           80000           Neukölln           436568
32 4           70000 Tempelhof-Schöneberg           398767
33 Command being timed: "python testcase2/prompt2/exec4/script.py"
34 User time (seconds): 4.66
35 System time (seconds): 0.24
36 Percent of CPU this job got: 163%
37 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:02.99
38 Average shared text size (kbytes): 0
39 Average unshared data size (kbytes): 0
40 Average stack size (kbytes): 0
41 Average total size (kbytes): 0
42 Maximum resident set size (kbytes): 153780
43 Average resident set size (kbytes): 0
44 Major (requiring I/O) page faults: 0
45 Minor (reclaiming a frame) page faults: 168577
46 Voluntary context switches: 33
47 Involuntary context switches: 257
```

```

48 Swaps: 0
49 File system inputs: 0
50 File system outputs: 0
51 Socket messages sent: 0
52 Socket messages received: 0
53 Signals delivered: 0
54 Page size (bytes): 4096
55 Exit status: 0

```

Listing 216: Ausgabe Testfall 2Prompt 2Ausführung 4

```

1 /home/mauke/Documents/Uni/Bachelorarbeit/testcases/testcase2/prompt2/exec5/
  script.py:43: SettingWithCopyWarning:
2 A value is trying to be set on a copy of a slice from a DataFrame.
3 Try using .loc[row_indexer,col_indexer] = value instead
4
5 See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
  stable/user_guide/indexing.html#returning-a-view-versus-a-copy
6 df_selected['Sheet'] = sheet_name
7 /home/mauke/Documents/Uni/Bachelorarbeit/testcases/testcase2/prompt2/exec5/
  script.py:43: SettingWithCopyWarning:
8 A value is trying to be set on a copy of a slice from a DataFrame.
9 Try using .loc[row_indexer,col_indexer] = value instead
10
11 See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
  stable/user_guide/indexing.html#returning-a-view-versus-a-copy
12 df_selected['Sheet'] = sheet_name
13 /home/mauke/Documents/Uni/Bachelorarbeit/testcases/testcase2/prompt2/exec5/
  script.py:43: SettingWithCopyWarning:
14 A value is trying to be set on a copy of a slice from a DataFrame.
15 Try using .loc[row_indexer,col_indexer] = value instead
16
17 See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
  stable/user_guide/indexing.html#returning-a-view-versus-a-copy
18 df_selected['Sheet'] = sheet_name
19 /home/mauke/Documents/Uni/Bachelorarbeit/testcases/testcase2/prompt2/exec5/
  script.py:43: SettingWithCopyWarning:
20 A value is trying to be set on a copy of a slice from a DataFrame.
21 Try using .loc[row_indexer,col_indexer] = value instead
22
23 See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
  stable/user_guide/indexing.html#returning-a-view-versus-a-copy
24 df_selected['Sheet'] = sheet_name
25 /home/mauke/Documents/Uni/Bachelorarbeit/testcases/testcase2/prompt2/exec5/
  script.py:43: SettingWithCopyWarning:
26 A value is trying to be set on a copy of a slice from a DataFrame.
27 Try using .loc[row_indexer,col_indexer] = value instead
28
29 See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
  stable/user_guide/indexing.html#returning-a-view-versus-a-copy
30 df_selected['Sheet'] = sheet_name
31 /home/mauke/Documents/Uni/Bachelorarbeit/testcases/testcase2/prompt2/exec5/
  script.py:43: SettingWithCopyWarning:

```

```

32 A value is trying to be set on a copy of a slice from a DataFrame.
33 Try using .loc[row_indexer,col_indexer] = value instead
34
35 See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
    stable/user_guide/indexing.html#returning-a-view-versus-a-copy
36 df_selected['Sheet'] = sheet_name
37 /home/mauke/Documents/Uni/Bachelorarbeit/testcases/testcase2/prompt2/exec5/
    script.py:43: SettingWithCopyWarning:
38 A value is trying to be set on a copy of a slice from a DataFrame.
39 Try using .loc[row_indexer,col_indexer] = value instead
40
41 See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
    stable/user_guide/indexing.html#returning-a-view-versus-a-copy
42 df_selected['Sheet'] = sheet_name
43 /home/mauke/Documents/Uni/Bachelorarbeit/testcases/testcase2/prompt2/exec5/
    script.py:43: SettingWithCopyWarning:
44 A value is trying to be set on a copy of a slice from a DataFrame.
45 Try using .loc[row_indexer,col_indexer] = value instead
46
47 See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
    stable/user_guide/indexing.html#returning-a-view-versus-a-copy
48 df_selected['Sheet'] = sheet_name
49 /home/mauke/Documents/Uni/Bachelorarbeit/testcases/testcase2/prompt2/exec5/
    script.py:43: SettingWithCopyWarning:
50 A value is trying to be set on a copy of a slice from a DataFrame.
51 Try using .loc[row_indexer,col_indexer] = value instead
52
53 See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
    stable/user_guide/indexing.html#returning-a-view-versus-a-copy
54 df_selected['Sheet'] = sheet_name
55 /home/mauke/Documents/Uni/Bachelorarbeit/testcases/testcase2/prompt2/exec5/
    script.py:43: SettingWithCopyWarning:
56 A value is trying to be set on a copy of a slice from a DataFrame.
57 Try using .loc[row_indexer,col_indexer] = value instead
58
59 See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
    stable/user_guide/indexing.html#returning-a-view-versus-a-copy
60 df_selected['Sheet'] = sheet_name
61 LOR-Schlüssel           Bezirke  Straftaten_insgesamt
62 0           10000           Mitte           878048
63 1           20000 Friedrichshain-Kreuzberg           579591
64 2           40000 Charlottenburg-Wilmersdorf           519706
65 3           80000           Neukölln           436568
66 4           70000 Tempelhof-Schöneberg           398767
67 5           30000           Pankow           393574
68 6           120000 Reinickendorf           284193
69 7           110000 Lichtenberg           279018
70 8           50000           Spandau           257757
71 9           90000 Treptow-Köpenick           247383
72 10          100000 Marzahn-Hellersdorf           241928
73 11           60000 Steglitz-Zehlendorf           241205
74 Command being timed: "python testcase2/prompt2/exec5/script.py"

```

```

75 User time (seconds): 4.65
76 System time (seconds): 0.22
77 Percent of CPU this job got: 165%
78 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:02.94
79 Average shared text size (kbytes): 0
80 Average unshared data size (kbytes): 0
81 Average stack size (kbytes): 0
82 Average total size (kbytes): 0
83 Maximum resident set size (kbytes): 155036
84 Average resident set size (kbytes): 0
85 Major (requiring I/O) page faults: 0
86 Minor (reclaiming a frame) page faults: 169052
87 Voluntary context switches: 38
88 Involuntary context switches: 643
89 Swaps: 0
90 File system inputs: 0
91 File system outputs: 0
92 Socket messages sent: 0
93 Socket messages received: 0
94 Signals delivered: 0
95 Page size (bytes): 4096
96 Exit status: 0

```

Listing 217: Ausgabe Testfall 2 Prompt 2 Ausführung 5

	LOR-Schlüssel		Bezirke	Straftaten_insgesamt
	Raub ...	Sachbeschädigung_insgesamt	Sachbeschädigung_Graffiti	
	Rauschgiftdelikte	Kieztaten		
2	0	10000	Mitte	878048
	8428 ...		58297	13343
	34835	162698		
3	1	20000	Friedrichshain-Kreuzberg	579591
	8286 ...		50327	11883
	47259	114395		
4	2	40000	Charlottenburg-Wilmersdorf	519706
	4559 ...		37870	8244
	11836	103582		
5	3	80000	Neukölln	436568
	4987 ...		40143	5873
	17797	100781		
6	4	70000	Tempelhof-Schöneberg	398767
	3587 ...		34963	7521
	11880	88085		
7	5	30000	Pankow	393574
	3469 ...		42713	12078
	7445	89662		
8	6	120000	Reinickendorf	284193
	2662 ...		24468	4365
	7986	64419		
9	7	110000	Lichtenberg	279018
	2475 ...		29462	6310
	5317	66356		
10	8	50000	Spandau	257757


```

11 9      2614  ...      5701      64128      23414      2910
      90000      Treptow-Köpenick      247383
      1979  ...      28092      6807
      5731      55239
12 10      100000      Marzahn-Hellersdorf      241928
      2423  ...      27969      5282
      4388      63928
13 11      60000      Steglitz-Zehlendorf      241205
      1764  ...      26485      6525
      4272      48793
14 12      999900 Stadtgebiet Berlin, nicht zuzuordnen      512158
      848  ...      13419      5295
      6234      19928
15 13      999999      Berlin (PKS gesamt)      5269896
      48081 ...      437622      96436
      170681      1041994
16
17 [14 rows x 19 columns]
18 Command being timed: "python testcase2/prompt3/exec1/script.py"
19 User time (seconds): 4.65
20 System time (seconds): 0.22
21 Percent of CPU this job got: 165%
22 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:02.94
23 Average shared text size (kbytes): 0
24 Average unshared data size (kbytes): 0
25 Average stack size (kbytes): 0
26 Average total size (kbytes): 0
27 Maximum resident set size (kbytes): 154160
28 Average resident set size (kbytes): 0
29 Major (requiring I/O) page faults: 0
30 Minor (reclaiming a frame) page faults: 166029
31 Voluntary context switches: 34
32 Involuntary context switches: 498
33 Swaps: 0
34 File system inputs: 0
35 File system outputs: 0
36 Socket messages sent: 0
37 Socket messages received: 0
38 Signals delivered: 0
39 Page size (bytes): 4096
40 Exit status: 0

```

Listing 218: Ausgabe Testfall 2Prompt 3Ausführung 1

```

1  LOR-Schlüssel      Bezirke  Straftaten_insgesamt  Raub
   Straßenraub ... Brandstiftung Sachbeschädigung_insgesamt Sachbeschä
   digung_Graffiti Rauschgiftdelikte Kieztaten
2  0      10000      Mitte      878048  8428
      4951  ...      1114      58297
      13343      34835      162698
3  1      20000      Friedrichshain-Kreuzberg      579591  8286
      5359  ...      966      50327

```

```

      11883      47259      114395
4 2      40000 Charlottenburg-Wilmersdorf      519706 4559
      2510 ...      829      37870
      8244      11836      103582
5 3      80000      Neukölln      436568 4987
      2775 ...      1193      40143
      5873      17797      100781
6 4      70000 Tempelhof-Schöneberg      398767 3587
      2105 ...      752      34963
      7521      11880      88085
7
8 [5 rows x 19 columns]
9 Command being timed: "python testcase2/prompt3/exec2/script.py"
10 User time (seconds): 4.55
11 System time (seconds): 0.22
12 Percent of CPU this job got: 168%
13 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:02.83
14 Average shared text size (kbytes): 0
15 Average unshared data size (kbytes): 0
16 Average stack size (kbytes): 0
17 Average total size (kbytes): 0
18 Maximum resident set size (kbytes): 153944
19 Average resident set size (kbytes): 0
20 Major (requiring I/O) page faults: 0
21 Minor (reclaiming a frame) page faults: 167997
22 Voluntary context switches: 35
23 Involuntary context switches: 480
24 Swaps: 0
25 File system inputs: 0
26 File system outputs: 0
27 Socket messages sent: 0
28 Socket messages received: 0
29 Signals delivered: 0
30 Page size (bytes): 4096
31 Exit status: 0

```

Listing 219: Ausgabe Testfall 2Prompt 3Ausführung 2

```

1      LOR-Schlüssel      Bezirke      Straftaten_insgesamt
      Raub ... Sachbeschädigung_insgesamt Sachbeschädigung_Graffiti
      Rauschgiftdelikte Kieztaten
2 0      10000      Mitte      878048
      8428 ...      58297      13343
      34835      162698
3 1      20000      Friedrichshain-Kreuzberg      579591
      8286 ...      50327      11883
      47259      114395
4 2      40000      Charlottenburg-Wilmersdorf      519706
      4559 ...      37870      8244
      11836      103582
5 3      80000      Neukölln      436568
      4987 ...      40143      5873
      17797      100781

```

6	4	70000		Tempelhof-Schöneberg	398767
		3587 ...		34963	7521
		11880	88085		
7	5	30000		Pankow	393574
		3469 ...		42713	12078
		7445	89662		
8	6	120000		Reinickendorf	284193
		2662 ...		24468	4365
		7986	64419		
9	7	110000		Lichtenberg	279018
		2475 ...		29462	6310
		5317	66356		
10	8	50000		Spandau	257757
		2614 ...		23414	2910
		5701	64128		
11	9	90000		Treptow-Köpenick	247383
		1979 ...		28092	6807
		5731	55239		
12	10	100000		Marzahn-Hellersdorf	241928
		2423 ...		27969	5282
		4388	63928		
13	11	60000		Steglitz-Zehlendorf	241205
		1764 ...		26485	6525
		4272	48793		
14	12	999900	Stadtgebiet Berlin, nicht zuzuordnen		512158
		848 ...		13419	5295
		6234	19928		
15	13	999999		Berlin (PKS gesamt)	5269896
		48081 ...		437622	96436
		170681	1041994		

16

17 [14 rows x 19 columns]

18 Command being timed: "python testcase2/prompt3/exec3/script.py"

19 User time (seconds): 4.95

20 System time (seconds): 0.23

21 Percent of CPU this job got: 157%

22 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:03.28

23 Average shared text size (kbytes): 0

24 Average unshared data size (kbytes): 0

25 Average stack size (kbytes): 0

26 Average total size (kbytes): 0

27 Maximum resident set size (kbytes): 153780

28 Average resident set size (kbytes): 0

29 Major (requiring I/O) page faults: 0

30 Minor (reclaiming a frame) page faults: 167786

31 Voluntary context switches: 34

32 Involuntary context switches: 500

33 Swaps: 0

34 File system inputs: 0

35 File system outputs: 0

36 Socket messages sent: 0

37 Socket messages received: 0

```

38 Signals delivered: 0
39 Page size (bytes): 4096
40 Exit status: 0

```

Listing 220: Ausgabe Testfall 2Prompt 3Ausführung 3

```

1  LOR-Schlüssel           Bezirke  Straftaten_insgesamt  Raub
   Straßenraub ... Brandstiftung Sachbeschädigung_insgesamt Sachbeschä
   digung_Graffiti Rauschgiftdelikte Kieztaten
2  0      10000           Mitte      878048  8428
   4951 ...      1114      58297
   13343      34835      162698
3  1      20000 Friedrichshain-Kreuzberg      579591  8286
   5359 ...      966      50327
   11883      47259      114395
4  2      40000 Charlottenburg-Wilmersdorf      519706  4559
   2510 ...      829      37870
   8244      11836      103582
5  3      80000           Neukölln      436568  4987
   2775 ...      1193      40143
   5873      17797      100781
6  4      70000 Tempelhof-Schöneberg      398767  3587
   2105 ...      752      34963
   7521      11880      88085
7
8  [5 rows x 19 columns]
9  Command being timed: "python testcase2/prompt3/exec4/script.py"
10 User time (seconds): 4.67
11 System time (seconds): 0.23
12 Percent of CPU this job got: 165%
13 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:02.96
14 Average shared text size (kbytes): 0
15 Average unshared data size (kbytes): 0
16 Average stack size (kbytes): 0
17 Average total size (kbytes): 0
18 Maximum resident set size (kbytes): 153820
19 Average resident set size (kbytes): 0
20 Major (requiring I/O) page faults: 0
21 Minor (reclaiming a frame) page faults: 167797
22 Voluntary context switches: 36
23 Involuntary context switches: 425
24 Swaps: 0
25 File system inputs: 0
26 File system outputs: 112
27 Socket messages sent: 0
28 Socket messages received: 0
29 Signals delivered: 0
30 Page size (bytes): 4096
31 Exit status: 0

```

Listing 221: Ausgabe Testfall 2Prompt 3Ausführung 4

1	LOR-Schlüssel	Bezirke	Straftaten_insgesamt
	Raub ... Sachbeschädigung_insgesamt Sachbeschädigung_Graffiti		
	Rauschgiftdelikte Kieztaten		
2	0	10000	Mitte
	1014 ...	5406	1177
	2880	15703	
3	1	20000	Friedrichshain-Kreuzberg
	940 ...	5034	1015
	3039	11082	
4	2	40000	Charlottenburg-Wilmersdorf
	641 ...	3422	792
	1208	10047	
5	3	80000	Neukölln
	580 ...	4253	746
	1180	9544	
6	4	70000	Tempelhof-Schöneberg
	496 ...	3367	651
	1146	8064	
7	5	30000	Pankow
	313 ...	3994	1036
	500	8463	
8	6	120000	Reinickendorf
	340 ...	2586	477
	829	7076	
9	7	50000	Spandau
	344 ...	2446	331
	549	6408	
10	8	110000	Lichtenberg
	236 ...	2508	559
	378	5708	
11	9	60000	Steglitz-Zehlendorf
	210 ...	2580	616
	433	4695	
12	10	100000	Marzahn-Hellersdorf
	260 ...	2565	448
	344	6181	
13	11	90000	Treptow-Köpenick
	209 ...	2517	576
	392	4729	
14	12	999900	Stadtgebiet Berlin, nicht zuzuordnen
	114 ...	2216	815
	587	2663	
15	13	999999	Berlin (PKS gesamt)
	5697 ...	42894	9239
	13465	100363	
16			
17	[14 rows x 19 columns]		
18	Command being timed: "python testcase2/prompt3/exec5/script.py"		
19	User time (seconds): 4.69		
20	System time (seconds): 0.23		
21	Percent of CPU this job got: 164%		
22	Elapsed (wall clock) time (h:mm:ss or m:ss): 0:02.99		

```

23 Average shared text size (kbytes): 0
24 Average unshared data size (kbytes): 0
25 Average stack size (kbytes): 0
26 Average total size (kbytes): 0
27 Maximum resident set size (kbytes): 154232
28 Average resident set size (kbytes): 0
29 Major (requiring I/O) page faults: 0
30 Minor (reclaiming a frame) page faults: 168184
31 Voluntary context switches: 39
32 Involuntary context switches: 542
33 Swaps: 0
34 File system inputs: 0
35 File system outputs: 0
36 Socket messages sent: 0
37 Socket messages received: 0
38 Signals delivered: 0
39 Page size (bytes): 4096
40 Exit status: 0

```

Listing 222: Ausgabe Testfall 2Prompt 3Ausführung 5

11.3.16 Komplette Ausgaben der Skripte Testfall 3

```

1 Die prozentualen Anteile wurden erfolgreich berechnet und in '
   Straftaten_Anteil_Bezirke.xlsx' gespeichert.
2 Command being timed: "python testcase3/prompt1/exec1/script.py"
3 User time (seconds): 2.44
4 System time (seconds): 0.07
5 Percent of CPU this job got: 444%
6 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.56
7 Average shared text size (kbytes): 0
8 Average unshared data size (kbytes): 0
9 Average stack size (kbytes): 0
10 Average total size (kbytes): 0
11 Maximum resident set size (kbytes): 153720
12 Average resident set size (kbytes): 0
13 Major (requiring I/O) page faults: 0
14 Minor (reclaiming a frame) page faults: 32451
15 Voluntary context switches: 34
16 Involuntary context switches: 384
17 Swaps: 0
18 File system inputs: 0
19 File system outputs: 80
20 Socket messages sent: 0
21 Socket messages received: 0
22 Signals delivered: 0
23 Page size (bytes): 4096
24 Exit status: 0

```

Listing 223: Ausgabe Testfall 3Prompt 1Ausführung 1

	Prozentualer_Anteil (%)	Bezirke	Straftaten_insgesamt
0	15.58	Mitte	83611
1	10.40	Friedrichshain-Kreuzberg	55838
2	7.30	Pankow	39164
3	9.99	Charlottenburg-Wilmersdorf	53630
4	4.76	Spandau	25564
5	4.42	Steglitz-Zehlendorf	23719
6	7.78	Tempelhof-Schöneberg	41776
7	8.44	Neukölln	45313
8	4.86	Treptow-Köpenick	26099
9	4.97	Marzahn-Hellersdorf	26649
10	5.88	Lichtenberg	31563
11	5.09	Reinickendorf	27344
12	10.51	Stadtgebiet Berlin, nicht zuzuordnen	56425
Die Ergebnisse wurden in 'Straftaten_Prozentual_Anteil.xlsx' gespeichert.			
Command being timed: "python testcase3/prompt1/exec2/script.py"			
User time (seconds): 2.45			
System time (seconds): 0.06			
Percent of CPU this job got: 446%			
Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.56			
Average shared text size (kbytes): 0			
Average unshared data size (kbytes): 0			
Average stack size (kbytes): 0			
Average total size (kbytes): 0			
Maximum resident set size (kbytes): 152956			
Average resident set size (kbytes): 0			
Major (requiring I/O) page faults: 0			
Minor (reclaiming a frame) page faults: 32504			
Voluntary context switches: 34			
Involuntary context switches: 536			
Swaps: 0			
File system inputs: 0			
File system outputs: 80			
Socket messages sent: 0			
Socket messages received: 0			
Signals delivered: 0			
Page size (bytes): 4096			

```
39 Exit status: 0
```

Listing 224: Ausgabe Testfall 3Prompt 1Ausführung 2

```
1           Bezirke   Straftaten_insgesamt   Prozentualer_Anteil (%)
2 0           Mitte           83611           15.58
3 1   Friedrichshain-Kreuzberg           55838           10.40
4 3   Charlottenburg-Wilmersdorf           53630           9.99
5 7           Neukölln           45313           8.44
6 6   Tempelhof-Schöneberg           41776           7.78
7 2           Pankow           39164           7.30
8 10          Lichtenberg           31563           5.88
9 11          Reinickendorf           27344           5.09
10 9   Marzahn-Hellersdorf           26649           4.97
11 8   Treptow-Köpenick           26099           4.86
12 4           Spandau           25564           4.76
13 5   Steglitz-Zehlendorf           23719           4.42
14 Command being timed: "python testcase3/prompt1/exec3/script.py"
15 User time (seconds): 2.41
16 System time (seconds): 0.07
17 Percent of CPU this job got: 442%
18 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.56
19 Average shared text size (kbytes): 0
20 Average unshared data size (kbytes): 0
21 Average stack size (kbytes): 0
22 Average total size (kbytes): 0
23 Maximum resident set size (kbytes): 153688
24 Average resident set size (kbytes): 0
25 Major (requiring I/O) page faults: 0
26 Minor (reclaiming a frame) page faults: 32482
27 Voluntary context switches: 39
28 Involuntary context switches: 211
29 Swaps: 0
30 File system inputs: 0
31 File system outputs: 0
32 Socket messages sent: 0
33 Socket messages received: 0
34 Signals delivered: 0
35 Page size (bytes): 4096
36 Exit status: 0
```

Listing 225: Ausgabe Testfall 3Prompt 1Ausführung 3

```
1 Prozentualer Anteil der Straftaten nach Bezirk:
2           Bezirke   Straftaten_insgesamt   Prozentualer_Anteil
3 0           Mitte           83611           15.58
4 1   Friedrichshain-Kreuzberg           55838           10.40
5 2           Pankow           39164           7.30
6 3   Charlottenburg-Wilmersdorf           53630           9.99
7 4           Spandau           25564           4.76
8 5   Steglitz-Zehlendorf           23719           4.42
9 6   Tempelhof-Schöneberg           41776           7.78
10 7           Neukölln           45313           8.44
```



```

11 8           Treptow-Köpenick           26099           4.86
12 9           Marzahn-Hellersdorf        26649           4.97
13 10          Lichtenberg                31563           5.88
14 11          Reinickendorf              27344           5.09
15
16 Die Ergebnisse wurden in 'Prozentuale_Anteile_Straftaten.xlsx' gespeichert.
17 Command being timed: "python testcase3/prompt1/exec4/script.py"
18 User time (seconds): 2.43
19 System time (seconds): 0.05
20 Percent of CPU this job got: 428%
21 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.58
22 Average shared text size (kbytes): 0
23 Average unshared data size (kbytes): 0
24 Average stack size (kbytes): 0
25 Average total size (kbytes): 0
26 Maximum resident set size (kbytes): 153660
27 Average resident set size (kbytes): 0
28 Major (requiring I/O) page faults: 0
29 Minor (reclaiming a frame) page faults: 32491
30 Voluntary context switches: 36
31 Involuntary context switches: 261
32 Swaps: 0
33 File system inputs: 0
34 File system outputs: 80
35 Socket messages sent: 0
36 Socket messages received: 0
37 Signals delivered: 0
38 Page size (bytes): 4096
39 Exit status: 0

```

Listing 226: Ausgabe Testfall 3Prompt 1Ausführung 4

```

1 Gesamte Straftaten in Berlin: 536695
2
3 Prozentualer Anteil der Straftaten pro Bezirk:
4
5      Bezirke   Straftaten_insgesamt   Prozentualer_Anteil (%)
6 0           Mitte                    83611                    15.58
7 1 Friedrichshain-Kreuzberg          55838                    10.40
8 2           Pankow                    39164                     7.30
9 3 Charlottenburg-Wilmersdorf        53630                     9.99
10 4           Spandau                    25564                     4.76
11 5 Steglitz-Zehlendorf                23719                     4.42
12 6 Tempelhof-Schöneberg               41776                     7.78
13 7           Neukölln                  45313                     8.44
14 8 Treptow-Köpenick                   26099                     4.86
15 9 Marzahn-Hellersdorf                26649                     4.97
16 10          Lichtenberg                31563                     5.88
17 11          Reinickendorf              27344                     5.09
18
19 Das Ergebnis wurde in 'Straftaten_Prozentual.xlsx' gespeichert.
20 Command being timed: "python testcase3/prompt1/exec5/script.py"
21 User time (seconds): 2.39
22 System time (seconds): 0.06

```

```

22 Percent of CPU this job got: 424%
23 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.58
24 Average shared text size (kbytes): 0
25 Average unshared data size (kbytes): 0
26 Average stack size (kbytes): 0
27 Average total size (kbytes): 0
28 Maximum resident set size (kbytes): 153332
29 Average resident set size (kbytes): 0
30 Major (requiring I/O) page faults: 0
31 Minor (reclaiming a frame) page faults: 32471
32 Voluntary context switches: 33
33 Involuntary context switches: 221
34 Swaps: 0
35 File system inputs: 0
36 File system outputs: 176
37 Socket messages sent: 0
38 Socket messages received: 0
39 Signals delivered: 0
40 Page size (bytes): 4096
41 Exit status: 0

```

Listing 227: Ausgabe Testfall 3Prompt 1Ausführung 5

	Prozent_Straftaten_insgesamt	Bezirke	Straftaten_insgesamt
0	15.58	Mitte	83611
1	10.40	Friedrichshain-Kreuzberg	55838
2	7.30	Pankow	39164
3	9.99	Charlottenburg-Wilmersdorf	53630
4	4.76	Spandau	25564
5	4.42	Steglitz-Zehlendorf	23719
6	7.78	Tempelhof-Schöneberg	41776
7	8.44	Neukölln	45313
8	4.86	Treptow-Köpenick	26099
9	4.97	Marzahn-Hellersdorf	26649
10	5.88	Lichtenberg	31563
11	5.09	Reinickendorf	27344
12	10.51	Stadtgebiet Berlin, nicht zuzuordnen	56425

Die Ergebnisse wurden in 'Fallzahlen_mit_Prozentsatz.xlsx' gespeichert.

```

17 Command being timed: "python testcase3/prompt2/exec1/script.py"
18 User time (seconds): 2.45
19 System time (seconds): 0.06
20 Percent of CPU this job got: 426%
21 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.59
22 Average shared text size (kbytes): 0
23 Average unshared data size (kbytes): 0
24 Average stack size (kbytes): 0
25 Average total size (kbytes): 0
26 Maximum resident set size (kbytes): 153468
27 Average resident set size (kbytes): 0
28 Major (requiring I/O) page faults: 0
29 Minor (reclaiming a frame) page faults: 32527
30 Voluntary context switches: 35
31 Involuntary context switches: 203
32 Swaps: 0
33 File system inputs: 0
34 File system outputs: 176
35 Socket messages sent: 0
36 Socket messages received: 0
37 Signals delivered: 0
38 Page size (bytes): 4096
39 Exit status: 0

```

Listing 228: Ausgabe Testfall 3Prompt 2Ausführung 1

	Prozentualer_Anteil (%)	Bezirke	Straftaten_insgesamt
0	15.58	Mitte	83611
12	10.51	Stadtgebiet Berlin, nicht zuzuordnen	56425
1	10.40	Friedrichshain-Kreuzberg	55838
3	9.99	Charlottenburg-Wilmersdorf	53630
7	8.44	Neukölln	45313
6	7.78	Tempelhof-Schöneberg	41776
2	7.30	Pankow	39164
10	5.88	Lichtenberg	31563
11	5.09	Reinickendorf	27344
9	4.97	Marzahn-Hellersdorf	26649
8	4.86	Treptow-Köpenick	26099
4	4.76	Spandau	25564
5		Steglitz-Zehlendorf	23719

```

15         4.42
16 Die Ergebnisse wurden in 'Fallzahlen_prozentual.xlsx' gespeichert.
17 Command being timed: "python testcase3/prompt2/exec2/script.py"
18 User time (seconds): 2.41
19 System time (seconds): 0.06
20 Percent of CPU this job got: 470%
21 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.52
22 Average shared text size (kbytes): 0
23 Average unshared data size (kbytes): 0
24 Average stack size (kbytes): 0
25 Average total size (kbytes): 0
26 Maximum resident set size (kbytes): 153352
27 Average resident set size (kbytes): 0
28 Major (requiring I/O) page faults: 0
29 Minor (reclaiming a frame) page faults: 32532
30 Voluntary context switches: 33
31 Involuntary context switches: 403
32 Swaps: 0
33 File system inputs: 0
34 File system outputs: 80
35 Socket messages sent: 0
36 Socket messages received: 0
37 Signals delivered: 0
38 Page size (bytes): 4096
39 Exit status: 0

```

Listing 229: Ausgabe Testfall 3Prompt 2Ausführung 2

	Prozentualer_Anteil	Bezirke	Straftaten_insgesamt
0	15.58	Mitte	83611
1	10.40	Friedrichshain-Kreuzberg	55838
2	7.30	Pankow	39164
3	9.99	Charlottenburg-Wilmersdorf	53630
4	4.76	Spandau	25564
5	4.42	Steglitz-Zehlendorf	23719
6	7.78	Tempelhof-Schöneberg	41776
7	8.44	Neukölln	45313
8	4.86	Treptow-Köpenick	26099
9	4.97	Marzahn-Hellersdorf	26649
10	5.88	Lichtenberg	31563

```

13 11                      Reinickendorf                      27344
    5.09
14 12 Stadtgebiet Berlin, nicht zuzuordnen                    56425
    10.51
15
16 Die prozentualen Anteile wurden in 'Fallzahlen_Prozente.xlsx' gespeichert.
17 Command being timed: "python testcase3/prompt2/exec3/script.py"
18 User time (seconds): 2.43
19 System time (seconds): 0.07
20 Percent of CPU this job got: 448%
21 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.55
22 Average shared text size (kbytes): 0
23 Average unshared data size (kbytes): 0
24 Average stack size (kbytes): 0
25 Average total size (kbytes): 0
26 Maximum resident set size (kbytes): 153716
27 Average resident set size (kbytes): 0
28 Major (requiring I/O) page faults: 0
29 Minor (reclaiming a frame) page faults: 32494
30 Voluntary context switches: 33
31 Involuntary context switches: 391
32 Swaps: 0
33 File system inputs: 0
34 File system outputs: 176
35 Socket messages sent: 0
36 Socket messages received: 0
37 Signals delivered: 0
38 Page size (bytes): 4096
39 Exit status: 0

```

Listing 230: Ausgabe Testfall 3Prompt 2Ausführung 3

	Prozentualer_Anteil	Bezirke	Straftaten_insgesamt
0	15.58	Mitte	83611
1	10.40	Friedrichshain-Kreuzberg	55838
2	7.30	Pankow	39164
3	9.99	Charlottenburg-Wilmersdorf	53630
4	4.76	Spandau	25564
5	4.42	Steglitz-Zehlendorf	23719
6	7.78	Tempelhof-Schöneberg	41776
7	8.44	Neukölln	45313
8	4.86	Treptow-Köpenick	26099
9		Marzahn-Hellersdorf	26649

```

12 10      4.97      Lichtenberg      31563
      5.88
13 11      Reinickendorf      27344
      5.09
14 12 Stadtgebiet Berlin, nicht zuzuordnen      56425
      10.51
15 Command being timed: "python testcase3/prompt2/exec4/script.py"
16 User time (seconds): 2.45
17 System time (seconds): 0.05
18 Percent of CPU this job got: 448%
19 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.56
20 Average shared text size (kbytes): 0
21 Average unshared data size (kbytes): 0
22 Average stack size (kbytes): 0
23 Average total size (kbytes): 0
24 Maximum resident set size (kbytes): 153720
25 Average resident set size (kbytes): 0
26 Major (requiring I/O) page faults: 0
27 Minor (reclaiming a frame) page faults: 32525
28 Voluntary context switches: 43
29 Involuntary context switches: 422
30 Swaps: 0
31 File system inputs: 0
32 File system outputs: 144
33 Socket messages sent: 0
34 Socket messages received: 0
35 Signals delivered: 0
36 Page size (bytes): 4096
37 Exit status: 0

```

Listing 231: Ausgabe Testfall 3Prompt 2Ausführung 4

	Prozent_Straftaten	Bezirke	Straftaten_insgesamt
2 0		Mitte	83611
	15.578867		
3 12		Stadtgebiet Berlin, nicht zuzuordnen	56425
	10.513420		
4 1		Friedrichshain-Kreuzberg	55838
	10.404047		
5 3		Charlottenburg-Wilmersdorf	53630
	9.992640		
6 7		Neukölln	45313
	8.442970		
7 6		Tempelhof-Schöneberg	41776
	7.783937		
8 2		Pankow	39164
	7.297254		
9 10		Lichtenberg	31563
	5.880994		
10 11		Reinickendorf	27344
	5.094886		

```

11 9 Marzahn-Hellersdorf 26649
    4.965390
12 8 Treptow-Köpenick 26099
    4.862911
13 4 Spandau 25564
    4.763227
14 5 Steglitz-Zehlendorf 23719
    4.419456
15
16 Die Ergebnisse wurden in 'Fallzahlen_mit_Prozent.xlsx' gespeichert.
17 Command being timed: "python testcase3/prompt2/exec5/script.py"
18 User time (seconds): 2.40
19 System time (seconds): 0.06
20 Percent of CPU this job got: 479%
21 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.51
22 Average shared text size (kbytes): 0
23 Average unshared data size (kbytes): 0
24 Average stack size (kbytes): 0
25 Average total size (kbytes): 0
26 Maximum resident set size (kbytes): 153784
27 Average resident set size (kbytes): 0
28 Major (requiring I/O) page faults: 0
29 Minor (reclaiming a frame) page faults: 32528
30 Voluntary context switches: 38
31 Involuntary context switches: 400
32 Swaps: 0
33 File system inputs: 0
34 File system outputs: 80
35 Socket messages sent: 0
36 Socket messages received: 0
37 Signals delivered: 0
38 Page size (bytes): 4096
39 Exit status: 0

```

Listing 232: Ausgabe Testfall 3Prompt 2Ausführung 5

```

1 Excel-Datei 'Fallzahlen.xlsx' erfolgreich eingelesen.
2 Alle erforderlichen Spalten sind vorhanden.
3 Gesamtzahl der Straftaten für Berlin: 536695
4 Prozentualer Anteil der Straftaten pro Bezirk berechnet.
5
6 Ergebnis DataFrame:
7
8      Bezirke  Straftaten_insgesamt
9      Prozentualer_Anteil (%)
10
11      Mitte  83611
12      15.58
13      Friedrichshain-Kreuzberg  55838
14      10.40
15      Pankow  39164
16      7.30
17      Charlottenburg-Wilmersdorf  53630
18      9.99
19      Spandau  25564

```

```

13      4.76      Steglitz-Zehlendorf      23719
14      4.42      Tempelhof-Schöneberg      41776
15      7.78      Neukölln      45313
16      8.44      Treptow-Köpenick      26099
17      4.86      Marzahn-Hellersdorf      26649
18      4.97      Lichtenberg      31563
19      5.88      Reinickendorf      27344
20      5.09      Stadtgebiet Berlin, nicht zuzuordnen      56425
21      10.51
22 Ergebnis erfolgreich in 'Fallzahlen_prozentual.xlsx' gespeichert.
23 Command being timed: "python testcase3/prompt3/exec1/script.py"
24 User time (seconds): 2.41
25 System time (seconds): 0.06
26 Percent of CPU this job got: 478%
27 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.51
28 Average shared text size (kbytes): 0
29 Average unshared data size (kbytes): 0
30 Average stack size (kbytes): 0
31 Average total size (kbytes): 0
32 Maximum resident set size (kbytes): 153276
33 Average resident set size (kbytes): 0
34 Major (requiring I/O) page faults: 0
35 Minor (reclaiming a frame) page faults: 32462
36 Voluntary context switches: 35
37 Involuntary context switches: 329
38 Swaps: 0
39 File system inputs: 0
40 File system outputs: 112
41 Socket messages sent: 0
42 Socket messages received: 0
43 Signals delivered: 0
44 Page size (bytes): 4096
45 Exit status: 0

```

Listing 233: Ausgabe Testfall 3Prompt 3Ausführung 1

```

1 Excel-Datei erfolgreich eingelesen.
2 Alle erforderlichen Spalten sind vorhanden.
3 Gesamtzahl der Straftaten in Berlin: 536695
4
5 Prozentualer Anteil der Straftaten pro Bezirk in Berlin:
6      Bezirke      Straftaten_insgesamt
7      Prozentualer_Anteil
8
9      Mitte      83611

```



```

15.58
8      Friedrichshain-Kreuzberg      55838
10.40
9      Pankow      39164
7.30
10     Charlottenburg-Wilmersdorf    53630
9.99
11     Spandau      25564
4.76
12     Steglitz-Zehlendorf      23719
4.42
13     Tempelhof-Schöneberg      41776
7.78
14     Neukölln      45313
8.44
15     Treptow-Köpenick      26099
4.86
16     Marzahn-Hellersdorf      26649
4.97
17     Lichtenberg      31563
5.88
18     Reinickendorf      27344
5.09
19 Stadtgebiet Berlin, nicht zuzuordnen 56425
10.51
20 Command being timed: "python testcase3/prompt3/exec2/script.py"
21 User time (seconds): 2.42
22 System time (seconds): 0.07
23 Percent of CPU this job got: 456%
24 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.54
25 Average shared text size (kbytes): 0
26 Average unshared data size (kbytes): 0
27 Average stack size (kbytes): 0
28 Average total size (kbytes): 0
29 Maximum resident set size (kbytes): 153656
30 Average resident set size (kbytes): 0
31 Major (requiring I/O) page faults: 0
32 Minor (reclaiming a frame) page faults: 32463
33 Voluntary context switches: 35
34 Involuntary context switches: 426
35 Swaps: 0
36 File system inputs: 0
37 File system outputs: 0
38 Socket messages sent: 0
39 Socket messages received: 0
40 Signals delivered: 0
41 Page size (bytes): 4096
42 Exit status: 0

```

Listing 234: Ausgabe Testfall 3Prompt 3Ausführung 2

```

1 Excel-Datei erfolgreich eingelesen.
2 Alle erforderlichen Spalten sind vorhanden.

```

```

3 Gesamtzahl der Straftaten für ganz Berlin: 536695
4 Prozentualer Anteil der Straftaten pro Bezirk berechnet.
5
6 Ergebnis:
7
8           Prozentualer_Anteil           Bezirke   Straftaten_insgesamt
9 0           15.58           Mitte           83611
10           10.40           Friedrichshain-Kreuzberg           55838
11           7.30           Pankow           39164
12           9.99           Charlottenburg-Wilmersdorf           53630
13           4.76           Spandau           25564
14           4.42           Steglitz-Zehlendorf           23719
15           7.78           Tempelhof-Schöneberg           41776
16           8.44           Neukölln           45313
17           4.86           Treptow-Köpenick           26099
18           4.97           Marzahn-Hellersdorf           26649
19           5.88           Lichtenberg           31563
20           5.09           Reinickendorf           27344
21 12 Stadtgebiet Berlin, nicht zuzuordnen           56425
22           10.51
23 Command being timed: "python testcase3/prompt3/exec3/script.py"
24 User time (seconds): 2.37
25 System time (seconds): 0.06
26 Percent of CPU this job got: 455%
27 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.53
28 Average shared text size (kbytes): 0
29 Average unshared data size (kbytes): 0
30 Average stack size (kbytes): 0
31 Average total size (kbytes): 0
32 Maximum resident set size (kbytes): 153620
33 Average resident set size (kbytes): 0
34 Major (requiring I/O) page faults: 0
35 Minor (reclaiming a frame) page faults: 32454
36 Voluntary context switches: 34
37 Involuntary context switches: 254
38 Swaps: 0
39 File system inputs: 0
40 File system outputs: 0
Socket messages sent: 0
Socket messages received: 0

```

```

41 Signals delivered: 0
42 Page size (bytes): 4096
43 Exit status: 0

```

Listing 235: Ausgabe Testfall 3Prompt 3Ausführung 3

```

1 File "/home/mauke/Documents/Uni/Bachelorarbeit/testcases/testcase3/prompt3/
  exec4/script.py", line 18
2     fehlende_spalten = [spalte for spalte in notwendige_spalten wenn spalte
  nicht in df.columns]
3
4 SyntaxError: invalid syntax
5 Command exited with non-zero status 1
6 Command being timed: "python testcase3/prompt3/exec4/script.py"
7 User time (seconds): 0.00
8 System time (seconds): 0.00
9 Percent of CPU this job got: 100%
10 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.01
11 Average shared text size (kbytes): 0
12 Average unshared data size (kbytes): 0
13 Average stack size (kbytes): 0
14 Average total size (kbytes): 0
15 Maximum resident set size (kbytes): 12700
16 Average resident set size (kbytes): 0
17 Major (requiring I/O) page faults: 0
18 Minor (reclaiming a frame) page faults: 1242
19 Voluntary context switches: 1
20 Involuntary context switches: 0
21 Swaps: 0
22 File system inputs: 0
23 File system outputs: 0
24 Socket messages sent: 0
25 Socket messages received: 0
26 Signals delivered: 0
27 Page size (bytes): 4096
28 Exit status: 1

```

Listing 236: Ausgabe Testfall 3Prompt 3Ausführung 4

```

1 Excel-Datei 'Fallzahlen.xlsx' erfolgreich eingelesen.
2 Überprüfung der notwendigen Spalten erfolgreich.
3 Gesamtzahl der Straftaten in Berlin: 536695
4
5 Prozentuale Anteile der Straftaten pro Bezirk:
6
7      Prozentualer_Anteil      Bezirke      Straftaten_insgesamt
8 0      15.58      Mitte      83611
9 1      10.40      Friedrichshain-Kreuzberg      55838
10 2      7.30      Pankow      39164

```

```

10 3      Charlottenburg-Wilmersdorf      53630
      9.99
11 4      Spandau      25564
      4.76
12 5      Steglitz-Zehlendorf      23719
      4.42
13 6      Tempelhof-Schöneberg      41776
      7.78
14 7      Neukölln      45313
      8.44
15 8      Treptow-Köpenick      26099
      4.86
16 9      Marzahn-Hellersdorf      26649
      4.97
17 10     Lichtenberg      31563
      5.88
18 11     Reinickendorf      27344
      5.09
19 12 Stadtgebiet Berlin, nicht zuzuordnen      56425
      10.51
20 Command being timed: "python testcase3/prompt3/exec5/script.py"
21 User time (seconds): 2.41
22 System time (seconds): 0.06
23 Percent of CPU this job got: 428%
24 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.57
25 Average shared text size (kbytes): 0
26 Average unshared data size (kbytes): 0
27 Average stack size (kbytes): 0
28 Average total size (kbytes): 0
29 Maximum resident set size (kbytes): 153632
30 Average resident set size (kbytes): 0
31 Major (requiring I/O) page faults: 0
32 Minor (reclaiming a frame) page faults: 32485
33 Voluntary context switches: 34
34 Involuntary context switches: 199
35 Swaps: 0
36 File system inputs: 0
37 File system outputs: 0
38 Socket messages sent: 0
39 Socket messages received: 0
40 Signals delivered: 0
41 Page size (bytes): 4096
42 Exit status: 0

```

Listing 237: Ausgabe Testfall 3Prompt 3Ausführung 5

11.3.17 Komplette Ausgaben der Skripte Testfall 4

	Straftaten_insgesamt	Prozentuale_Veraenderung
Jahr		
2014	543156	NaN
2015	569549	4.86

```

5 2016          568860          -0.12
6 2017          520437          -8.51
7 2018          511677          -1.68
8 2019          513426           0.34
9 2020          504142          -1.81
10 2021          482127          -4.37
11 2022          519827           7.82
12 2023          536695           3.24
13 Command being timed: "python testcase4/prompt1/exec1/script.py"
14 User time (seconds): 4.58
15 System time (seconds): 0.22
16 Percent of CPU this job got: 168%
17 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:02.86
18 Average shared text size (kbytes): 0
19 Average unshared data size (kbytes): 0
20 Average stack size (kbytes): 0
21 Average total size (kbytes): 0
22 Maximum resident set size (kbytes): 157068
23 Average resident set size (kbytes): 0
24 Major (requiring I/O) page faults: 0
25 Minor (reclaiming a frame) page faults: 169809
26 Voluntary context switches: 34
27 Involuntary context switches: 572
28 Swaps: 0
29 File system inputs: 0
30 File system outputs: 0
31 Socket messages sent: 0
32 Socket messages received: 0
33 Signals delivered: 0
34 Page size (bytes): 4096
35 Exit status: 0

```

Listing 238: Ausgabe Testfall 4Prompt 1Ausführung 1

```

1 Sheet 'Fallzahlen_2014' entspricht nicht dem erwarteten Jahresformat und wird
  übersprungen.
2 Sheet 'Fallzahlen_2015' entspricht nicht dem erwarteten Jahresformat und wird
  übersprungen.
3 Sheet 'Fallzahlen_2016' entspricht nicht dem erwarteten Jahresformat und wird
  übersprungen.
4 Sheet 'Fallzahlen_2017' entspricht nicht dem erwarteten Jahresformat und wird
  übersprungen.
5 Sheet 'Fallzahlen_2018' entspricht nicht dem erwarteten Jahresformat und wird
  übersprungen.
6 Sheet 'Fallzahlen_2019' entspricht nicht dem erwarteten Jahresformat und wird
  übersprungen.
7 Sheet 'Fallzahlen_2020' entspricht nicht dem erwarteten Jahresformat und wird
  übersprungen.
8 Sheet 'Fallzahlen_2021' entspricht nicht dem erwarteten Jahresformat und wird
  übersprungen.
9 Sheet 'Fallzahlen_2022' entspricht nicht dem erwarteten Jahresformat und wird
  übersprungen.
10 Sheet 'Fallzahlen_2023' entspricht nicht dem erwarteten Jahresformat und wird

```

```

    übersprungen.
11 Keine gültigen Daten gefunden.
12 Command being timed: "python testcase4/prompt1/exec2/script.py"
13 User time (seconds): 4.84
14 System time (seconds): 0.21
15 Percent of CPU this job got: 159%
16 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:03.16
17 Average shared text size (kbytes): 0
18 Average unshared data size (kbytes): 0
19 Average stack size (kbytes): 0
20 Average total size (kbytes): 0
21 Maximum resident set size (kbytes): 157716
22 Average resident set size (kbytes): 0
23 Major (requiring I/O) page faults: 0
24 Minor (reclaiming a frame) page faults: 170264
25 Voluntary context switches: 34
26 Involuntary context switches: 334
27 Swaps: 0
28 File system inputs: 0
29 File system outputs: 0
30 Socket messages sent: 0
31 Socket messages received: 0
32 Signals delivered: 0
33 Page size (bytes): 4096
34 Exit status: 0

```

Listing 239: Ausgabe Testfall 4Prompt 1Ausführung 2

```

1 File "/home/mauke/Documents/Uni/Bachelorarbeit/testcases/testcase4/prompt1/
  exec3/script.py", line 24
2     if nicht berlin_gesamt.empty:
3         ~~~~~
4 SyntaxError: invalid syntax
5 Command exited with non-zero status 1
6 Command being timed: "python testcase4/prompt1/exec3/script.py"
7 User time (seconds): 0.01
8 System time (seconds): 0.00
9 Percent of CPU this job got: 84%
10 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.01
11 Average shared text size (kbytes): 0
12 Average unshared data size (kbytes): 0
13 Average stack size (kbytes): 0
14 Average total size (kbytes): 0
15 Maximum resident set size (kbytes): 12660
16 Average resident set size (kbytes): 0
17 Major (requiring I/O) page faults: 0
18 Minor (reclaiming a frame) page faults: 1244
19 Voluntary context switches: 1
20 Involuntary context switches: 0
21 Swaps: 0
22 File system inputs: 0
23 File system outputs: 0
24 Socket messages sent: 0

```

```
25 Socket messages received: 0
26 Signals delivered: 0
27 Page size (bytes): 4096
28 Exit status: 1
```

Listing 240: Ausgabe Testfall 4Prompt 1Ausführung 3

```
1 /home/mauke/Documents/Uni/Bachelorarbeit/testcases/testcase4/prompt1/exec4/
  script.py:23: SyntaxWarning: invalid escape sequence '\('
2   total_row = df[df['LÖR-Schlüssel'].astype(str).str.contains('Berlin \((PKS
    gesamt\)'), regex=True)]
3 Warnung: Jahr konnte nicht aus dem Sheet-Namen 'Fallzahlen_2014' extrahiert
  werden. Überspringe dieses Sheet.
4 Warnung: Jahr konnte nicht aus dem Sheet-Namen 'Fallzahlen_2015' extrahiert
  werden. Überspringe dieses Sheet.
5 Warnung: Jahr konnte nicht aus dem Sheet-Namen 'Fallzahlen_2016' extrahiert
  werden. Überspringe dieses Sheet.
6 Warnung: Jahr konnte nicht aus dem Sheet-Namen 'Fallzahlen_2017' extrahiert
  werden. Überspringe dieses Sheet.
7 Warnung: Jahr konnte nicht aus dem Sheet-Namen 'Fallzahlen_2018' extrahiert
  werden. Überspringe dieses Sheet.
8 Warnung: Jahr konnte nicht aus dem Sheet-Namen 'Fallzahlen_2019' extrahiert
  werden. Überspringe dieses Sheet.
9 Warnung: Jahr konnte nicht aus dem Sheet-Namen 'Fallzahlen_2020' extrahiert
  werden. Überspringe dieses Sheet.
10 Warnung: Jahr konnte nicht aus dem Sheet-Namen 'Fallzahlen_2021' extrahiert
  werden. Überspringe dieses Sheet.
11 Warnung: Jahr konnte nicht aus dem Sheet-Namen 'Fallzahlen_2022' extrahiert
  werden. Überspringe dieses Sheet.
12 Warnung: Jahr konnte nicht aus dem Sheet-Namen 'Fallzahlen_2023' extrahiert
  werden. Überspringe dieses Sheet.
13 Traceback (most recent call last):
14   File "/home/mauke/Documents/Uni/Bachelorarbeit/testcases/testcase4/prompt1/
    exec4/script.py", line 53, in <module>
15     df_total = df_total.sort_values('Jahr').reset_index(drop=True)
16     ~~~~~
17   File "/home/mauke/Documents/Uni/Bachelorarbeit/venv/lib/python3.13/site-
    packages/pandas/core/frame.py", line 7189, in sort_values
18     k = self._get_label_or_level_values(by[0], axis=axis)
19   File "/home/mauke/Documents/Uni/Bachelorarbeit/venv/lib/python3.13/site-
    packages/pandas/core/generic.py", line 1911, in _get_label_or_level_values
20     raise KeyError(key)
21 KeyError: 'Jahr'
22 Command exited with non-zero status 1
23 Command being timed: "python testcase4/prompt1/exec4/script.py"
24 User time (seconds): 4.57
25 System time (seconds): 0.23
26 Percent of CPU this job got: 164%
27 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:02.93
28 Average shared text size (kbytes): 0
29 Average unshared data size (kbytes): 0
30 Average stack size (kbytes): 0
31 Average total size (kbytes): 0
```

```

32 Maximum resident set size (kbytes): 155724
33 Average resident set size (kbytes): 0
34 Major (requiring I/O) page faults: 0
35 Minor (reclaiming a frame) page faults: 168787
36 Voluntary context switches: 34
37 Involuntary context switches: 421
38 Swaps: 0
39 File system inputs: 0
40 File system outputs: 0
41 Socket messages sent: 0
42 Socket messages received: 0
43 Signals delivered: 0
44 Page size (bytes): 4096
45 Exit status: 1

```

Listing 241: Ausgabe Testfall 4Prompt 1Ausführung 4

```

1          Straftaten_insgesamt  % Veränderung Vorjahr
2 Jahr
3 Fallzahlen_2014          543156          NaN
4 Fallzahlen_2015          569549          4.86
5 Fallzahlen_2016          568860         -0.12
6 Fallzahlen_2017          520437         -8.51
7 Fallzahlen_2018          511677         -1.68
8 Fallzahlen_2019          513426          0.34
9 Fallzahlen_2020          504142         -1.81
10 Fallzahlen_2021          482127         -4.37
11 Fallzahlen_2022          519827          7.82
12 Fallzahlen_2023          536695          3.24
13 Command being timed: "python testcase4/prompt1/exec5/script.py"
14 User time (seconds): 4.66
15 System time (seconds): 0.23
16 Percent of CPU this job got: 165%
17 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:02.95
18 Average shared text size (kbytes): 0
19 Average unshared data size (kbytes): 0
20 Average stack size (kbytes): 0
21 Average total size (kbytes): 0
22 Maximum resident set size (kbytes): 157336
23 Average resident set size (kbytes): 0
24 Major (requiring I/O) page faults: 0
25 Minor (reclaiming a frame) page faults: 168939
26 Voluntary context switches: 40
27 Involuntary context switches: 503
28 Swaps: 0
29 File system inputs: 0
30 File system outputs: 80
31 Socket messages sent: 0
32 Socket messages received: 0
33 Signals delivered: 0
34 Page size (bytes): 4096
35 Exit status: 0

```


Listing 242: Ausgabe Testfall 4Prompt 1Ausführung 5

```

1      Year  Straftaten_insgesamt  Prozentuale_Veraenderung
2  0  2014                      543156                      NaN
3  1  2015                      569549                      4.86
4  2  2016                      568860                     -0.12
5  3  2017                      520437                     -8.51
6  4  2018                      511677                     -1.68
7  5  2019                      513426                      0.34
8  6  2020                      504142                     -1.81
9  7  2021                      482127                     -4.37
10 8  2022                      519827                      7.82
11 9  2023                      536695                      3.24
12 Command being timed: "python testcase4/prompt2/exec1/script.py"
13 User time (seconds): 4.70
14 System time (seconds): 0.27
15 Percent of CPU this job got: 162%
16 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:03.07
17 Average shared text size (kbytes): 0
18 Average unshared data size (kbytes): 0
19 Average stack size (kbytes): 0
20 Average total size (kbytes): 0
21 Maximum resident set size (kbytes): 159292
22 Average resident set size (kbytes): 0
23 Major (requiring I/O) page faults: 0
24 Minor (reclaiming a frame) page faults: 169805
25 Voluntary context switches: 33
26 Involuntary context switches: 477
27 Swaps: 0
28 File system inputs: 0
29 File system outputs: 0
30 Socket messages sent: 0
31 Socket messages received: 0
32 Signals delivered: 0
33 Page size (bytes): 4096
34 Exit status: 0

```

Listing 243: Ausgabe Testfall 4Prompt 2Ausführung 1

```

1 Prozentuale Veränderung der Straftaten_insgesamt zum Vorjahr:
2      Year  Percentage_Change
3  0  2014                      NaN
4  1  2015                      4.859193
5  2  2016                     -0.120973
6  3  2017                     -8.512288
7  4  2018                     -1.683201
8  5  2019                      0.341817
9  6  2020                     -1.808245
10 7  2021                     -4.366825
11 8  2022                      7.819516
12 9  2023                      3.244926

```

```

13 Command being timed: "python testcase4/prompt2/exec2/script.py"
14 User time (seconds): 4.72
15 System time (seconds): 0.24
16 Percent of CPU this job got: 161%
17 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:03.07
18 Average shared text size (kbytes): 0
19 Average unshared data size (kbytes): 0
20 Average stack size (kbytes): 0
21 Average total size (kbytes): 0
22 Maximum resident set size (kbytes): 158432
23 Average resident set size (kbytes): 0
24 Major (requiring I/O) page faults: 0
25 Minor (reclaiming a frame) page faults: 169006
26 Voluntary context switches: 33
27 Involuntary context switches: 538
28 Swaps: 0
29 File system inputs: 0
30 File system outputs: 0
31 Socket messages sent: 0
32 Socket messages received: 0
33 Signals delivered: 0
34 Page size (bytes): 4096
35 Exit status: 0

```

Listing 244: Ausgabe Testfall 4Prompt 2Ausführung 2

	Jahr	Straftaten_insgesamt	Prozentuale_Veränderung (%)
0	2014	543156	NaN
1	2015	569549	4.86
2	2016	568860	-0.12
3	2017	520437	-8.51
4	2018	511677	-1.68
5	2019	513426	0.34
6	2020	504142	-1.81
7	2021	482127	-4.37
8	2022	519827	7.82
9	2023	536695	3.24

```

12 Command being timed: "python testcase4/prompt2/exec3/script.py"
13 User time (seconds): 5.17
14 System time (seconds): 0.24
15 Percent of CPU this job got: 155%
16 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:03.47
17 Average shared text size (kbytes): 0
18 Average unshared data size (kbytes): 0
19 Average stack size (kbytes): 0
20 Average total size (kbytes): 0
21 Maximum resident set size (kbytes): 156816
22 Average resident set size (kbytes): 0
23 Major (requiring I/O) page faults: 0
24 Minor (reclaiming a frame) page faults: 168996
25 Voluntary context switches: 39
26 Involuntary context switches: 451
27 Swaps: 0

```

```

28 File system inputs: 0
29 File system outputs: 80
30 Socket messages sent: 0
31 Socket messages received: 0
32 Signals delivered: 0
33 Page size (bytes): 4096
34 Exit status: 0

```

Listing 245: Ausgabe Testfall 4Prompt 2Ausführung 3

```

1   Year  Straftaten_insgesamt  Percent  Change
2 0  2014                543156.0          NaN
3 1  2015                569549.0           4.86
4 2  2016                568860.0          -0.12
5 3  2017                520437.0          -8.51
6 4  2018                511677.0          -1.68
7 5  2019                513426.0           0.34
8 6  2020                504142.0          -1.81
9 7  2021                482127.0          -4.37
10 8  2022                519827.0           7.82
11 9  2023                536695.0           3.24
12 Command being timed: "python testcase4/prompt2/exec4/script.py"
13 User time (seconds): 4.53
14 System time (seconds): 0.24
15 Percent of CPU this job got: 167%
16 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:02.85
17 Average shared text size (kbytes): 0
18 Average unshared data size (kbytes): 0
19 Average stack size (kbytes): 0
20 Average total size (kbytes): 0
21 Maximum resident set size (kbytes): 157516
22 Average resident set size (kbytes): 0
23 Major (requiring I/O) page faults: 0
24 Minor (reclaiming a frame) page faults: 169008
25 Voluntary context switches: 39
26 Involuntary context switches: 611
27 Swaps: 0
28 File system inputs: 0
29 File system outputs: 0
30 Socket messages sent: 0
31 Socket messages received: 0
32 Signals delivered: 0
33 Page size (bytes): 4096
34 Exit status: 0

```

Listing 246: Ausgabe Testfall 4Prompt 2Ausführung 4

	Jahr	Straftaten_insgesamt	Prozentuale_Veraenderung
0	2014	543156	NaN
1	2015	569549	4.86
2	2016	568860	-0.12
3	2017	520437	-8.51
4	2018	511677	-1.68

```

7 5 2019 513426 0.34
8 6 2020 504142 -1.81
9 7 2021 482127 -4.37
10 8 2022 519827 7.82
11 9 2023 536695 3.24
12 Command being timed: "python testcase4/prompt2/exec5/script.py"
13 User time (seconds): 4.64
14 System time (seconds): 0.22
15 Percent of CPU this job got: 163%
16 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:02.97
17 Average shared text size (kbytes): 0
18 Average unshared data size (kbytes): 0
19 Average stack size (kbytes): 0
20 Average total size (kbytes): 0
21 Maximum resident set size (kbytes): 158020
22 Average resident set size (kbytes): 0
23 Major (requiring I/O) page faults: 0
24 Minor (reclaiming a frame) page faults: 170160
25 Voluntary context switches: 34
26 Involuntary context switches: 422
27 Swaps: 0
28 File system inputs: 0
29 File system outputs: 0
30 Socket messages sent: 0
31 Socket messages received: 0
32 Signals delivered: 0
33 Page size (bytes): 4096
34 Exit status: 0

```

Listing 247: Ausgabe Testfall 4Prompt 2Ausführung 5

```

1 Jahr Prozentuale Veränderung (%)
2 0 2015 4.86
3 1 2016 -0.12
4 2 2017 -8.51
5 3 2018 -1.68
6 4 2019 0.34
7 5 2020 -1.81
8 6 2021 -4.37
9 7 2022 7.82
10 8 2023 3.24
11 Command being timed: "python testcase4/prompt3/exec1/script.py"
12 User time (seconds): 4.78
13 System time (seconds): 0.23
14 Percent of CPU this job got: 162%
15 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:03.08
16 Average shared text size (kbytes): 0
17 Average unshared data size (kbytes): 0
18 Average stack size (kbytes): 0
19 Average total size (kbytes): 0
20 Maximum resident set size (kbytes): 156244
21 Average resident set size (kbytes): 0
22 Major (requiring I/O) page faults: 0

```

```

23 Minor (reclaiming a frame) page faults: 168789
24 Voluntary context switches: 33
25 Involuntary context switches: 453
26 Swaps: 0
27 File system inputs: 0
28 File system outputs: 80
29 Socket messages sent: 0
30 Socket messages received: 0
31 Signals delivered: 0
32 Page size (bytes): 4096
33 Exit status: 0

```

Listing 248: Ausgabe Testfall 4Prompt 3Ausführung 1

```

1   Jahr  Prozentuale_Veraenderung
2 0  2014                               NaN
3 1  2015                               4.86
4 2  2016                               -0.12
5 3  2017                               -8.51
6 4  2018                               -1.68
7 5  2019                               0.34
8 6  2020                               -1.81
9 7  2021                               -4.37
10 8  2022                               7.82
11 9  2023                               3.24
12 Command being timed: "python testcase4/prompt3/exec2/script.py"
13 User time (seconds): 4.61
14 System time (seconds): 0.26
15 Percent of CPU this job got: 166%
16 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:02.93
17 Average shared text size (kbytes): 0
18 Average unshared data size (kbytes): 0
19 Average stack size (kbytes): 0
20 Average total size (kbytes): 0
21 Maximum resident set size (kbytes): 156732
22 Average resident set size (kbytes): 0
23 Major (requiring I/O) page faults: 0
24 Minor (reclaiming a frame) page faults: 169039
25 Voluntary context switches: 36
26 Involuntary context switches: 457
27 Swaps: 0
28 File system inputs: 0
29 File system outputs: 0
30 Socket messages sent: 0
31 Socket messages received: 0
32 Signals delivered: 0
33 Page size (bytes): 4096
34 Exit status: 0

```

Listing 249: Ausgabe Testfall 4Prompt 3Ausführung 2

```

1   Jahr  Straftaten_insgesamt  Prozentuale_Veraenderung
2 0  2014                    543156                             NaN

```

```

3 1 2015 569549 4.86
4 2 2016 568860 -0.12
5 3 2017 520437 -8.51
6 4 2018 511677 -1.68
7 5 2019 513426 0.34
8 6 2020 504142 -1.81
9 7 2021 482127 -4.37
10 8 2022 519827 7.82
11 9 2023 536695 3.24
12 Ergebnisse wurden in 'Straftaten_Veraenderung.xlsx' gespeichert.
13 Command being timed: "python testcase4/prompt3/exec3/script.py"
14 User time (seconds): 4.77
15 System time (seconds): 0.24
16 Percent of CPU this job got: 162%
17 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:03.08
18 Average shared text size (kbytes): 0
19 Average unshared data size (kbytes): 0
20 Average stack size (kbytes): 0
21 Average total size (kbytes): 0
22 Maximum resident set size (kbytes): 156520
23 Average resident set size (kbytes): 0
24 Major (requiring I/O) page faults: 0
25 Minor (reclaiming a frame) page faults: 168933
26 Voluntary context switches: 34
27 Involuntary context switches: 578
28 Swaps: 0
29 File system inputs: 0
30 File system outputs: 176
31 Socket messages sent: 0
32 Socket messages received: 0
33 Signals delivered: 0
34 Page size (bytes): 4096
35 Exit status: 0

```

Listing 250: Ausgabe Testfall 4Prompt 3Ausführung 3

```

1 /home/mauke/Documents/Uni/Bachelorarbeit/testcases/testcase4/prompt3/exec4/
  script.py:55: FutureWarning: The behavior of DataFrame concatenation with
  empty or all-NA entries is deprecated. In a future version, this will no
  longer exclude empty or all-NA columns when determining the result dtypes.
  To retain the old behavior, exclude the relevant entries before the
  concat operation.
2 result_df = pd.concat([
3     Jahr  Prozentuale Veränderung
4 0 2014 NaN
5 1 2015 4.859193
6 2 2016 -0.120973
7 3 2017 -8.512288
8 4 2018 -1.683201
9 5 2019 0.341817
10 6 2020 -1.808245
11 7 2021 -4.366825
12 8 2022 7.819516

```

```

13 9 2023 3.244926
14 Command being timed: "python testcase4/prompt3/exec4/script.py"
15 User time (seconds): 4.98
16 System time (seconds): 0.24
17 Percent of CPU this job got: 158%
18 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:03.30
19 Average shared text size (kbytes): 0
20 Average unshared data size (kbytes): 0
21 Average stack size (kbytes): 0
22 Average total size (kbytes): 0
23 Maximum resident set size (kbytes): 157784
24 Average resident set size (kbytes): 0
25 Major (requiring I/O) page faults: 0
26 Minor (reclaiming a frame) page faults: 169522
27 Voluntary context switches: 35
28 Involuntary context switches: 552
29 Swaps: 0
30 File system inputs: 0
31 File system outputs: 80
32 Socket messages sent: 0
33 Socket messages received: 0
34 Signals delivered: 0
35 Page size (bytes): 4096
36 Exit status: 0

```

Listing 251: Ausgabe Testfall 4Prompt 3Ausführung 4

```

1 Jahr Prozentuale Veränderung (%)
2 0 2014 NaN
3 1 2015 4.86
4 2 2016 -0.12
5 3 2017 -8.51
6 4 2018 -1.68
7 5 2019 0.34
8 6 2020 -1.81
9 7 2021 -4.37
10 8 2022 7.82
11 9 2023 3.24
12 Command being timed: "python testcase4/prompt3/exec5/script.py"
13 User time (seconds): 4.76
14 System time (seconds): 0.23
15 Percent of CPU this job got: 160%
16 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:03.11
17 Average shared text size (kbytes): 0
18 Average unshared data size (kbytes): 0
19 Average stack size (kbytes): 0
20 Average total size (kbytes): 0
21 Maximum resident set size (kbytes): 156004
22 Average resident set size (kbytes): 0
23 Major (requiring I/O) page faults: 0
24 Minor (reclaiming a frame) page faults: 168952
25 Voluntary context switches: 33
26 Involuntary context switches: 436

```

```

27 Swaps: 0
28 File system inputs: 0
29 File system outputs: 80
30 Socket messages sent: 0
31 Socket messages received: 0
32 Signals delivered: 0
33 Page size (bytes): 4096
34 Exit status: 0

```

Listing 252: Ausgabe Testfall 4Prompt 3Ausführung 5

11.3.18 Komplette Ausgaben der Skripte Testfall 5

```

1 Unterbezirk mit den meisten Raubdelikten pro Oberbezirk (2023):
2
3 Oberbezirk: Charlottenburg-Wilmersdorf
4   Unterbezirk: Charlottenburg-Wilmersdorf
5   Anzahl Raubdelikte: 464
6
7 Oberbezirk: Friedrichshain-Kreuzberg
8   Unterbezirk: Friedrichshain-Kreuzberg
9   Anzahl Raubdelikte: 923
10
11 Oberbezirk: Lichtenberg
12   Unterbezirk: Lichtenberg
13   Anzahl Raubdelikte: 362
14
15 Oberbezirk: Marzahn-Hellersdorf
16   Unterbezirk: Marzahn-Hellersdorf
17   Anzahl Raubdelikte: 315
18
19 Oberbezirk: Mitte
20   Unterbezirk: Mitte
21   Anzahl Raubdelikte: 884
22
23 Oberbezirk: Neukölln
24   Unterbezirk: Neukölln
25   Anzahl Raubdelikte: 508
26
27 Oberbezirk: Pankow
28   Unterbezirk: Pankow
29   Anzahl Raubdelikte: 529
30
31 Oberbezirk: Reinickendorf
32   Unterbezirk: Reinickendorf
33   Anzahl Raubdelikte: 311
34
35 Oberbezirk: Spandau
36   Unterbezirk: Spandau
37   Anzahl Raubdelikte: 247
38
39 Oberbezirk: Steglitz-Zehlendorf

```



```

40 Unterbezirk: Steglitz-Zehlendorf
41 Anzahl Raubdelikte: 213
42
43 Oberbezirk: Tempelhof-Schöneberg
44 Unterbezirk: Tempelhof-Schöneberg
45 Anzahl Raubdelikte: 328
46
47 Oberbezirk: Treptow-Köpenick
48 Unterbezirk: Treptow-Köpenick
49 Anzahl Raubdelikte: 207
50
51 Command being timed: "python testcase5/prompt1/exec1/script.py"
52 User time (seconds): 2.41
53 System time (seconds): 0.06
54 Percent of CPU this job got: 433%
55 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.57
56 Average shared text size (kbytes): 0
57 Average unshared data size (kbytes): 0
58 Average stack size (kbytes): 0
59 Average total size (kbytes): 0
60 Maximum resident set size (kbytes): 156304
61 Average resident set size (kbytes): 0
62 Major (requiring I/O) page faults: 0
63 Minor (reclaiming a frame) page faults: 33032
64 Voluntary context switches: 39
65 Involuntary context switches: 336
66 Swaps: 0
67 File system inputs: 0
68 File system outputs: 0
69 Socket messages sent: 0
70 Socket messages received: 0
71 Signals delivered: 0
72 Page size (bytes): 4096
73 Exit status: 0

```

Listing 253: Ausgabe Testfall 5Prompt 1Ausführung 1

```

1 Warnung: Einige Oberbezirk-Codes konnten nicht zugeordnet werden: ['20' '21'
  '22' '23' '24' '25' '29' '30' '31' '32' '33' '34' '35' '36'
2 '37' '39' '40' '41' '42' '43' '44' '45' '49' '50' '51' '52' '53' '54'
3 '59' '60' '61' '62' '63' '64' '69' '70' '71' '72' '73' '74' '75' '76'
4 '79' '80' '81' '82' '83' '84' '89' '90' '91' '92' '93' '94' '95' '99']
5 Unterbezirk mit den meisten Raubdelikten pro Oberbezirk:
6
7 Oberbezirk Bezeichnung (Bezirksregion) Raub
8 Charlottenburg-Wilmersdorf Reinickendorf 311
9 Friedrichshain-Kreuzberg Osloer Straße 89
10 Lichtenberg Wedding Zentrum 166
11 Mitte Mitte 884
12 Pankow Lichtenberg 362
13 Steglitz-Zehlendorf Bezirk (Mi), nicht zuzuordnen 28
14
15 Das Ergebnis wurde in 'Meiste_Raubdelikte_pro_Oberbezirk.xlsx' gespeichert.
16 Command being timed: "python testcase5/prompt1/exec2/script.py"

```

```

16 User time (seconds): 2.39
17 System time (seconds): 0.06
18 Percent of CPU this job got: 438%
19 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.56
20 Average shared text size (kbytes): 0
21 Average unshared data size (kbytes): 0
22 Average stack size (kbytes): 0
23 Average total size (kbytes): 0
24 Maximum resident set size (kbytes): 156744
25 Average resident set size (kbytes): 0
26 Major (requiring I/O) page faults: 0
27 Minor (reclaiming a frame) page faults: 33178
28 Voluntary context switches: 47
29 Involuntary context switches: 254
30 Swaps: 0
31 File system inputs: 72
32 File system outputs: 144
33 Socket messages sent: 0
34 Socket messages received: 0
35 Signals delivered: 0
36 Page size (bytes): 4096
37 Exit status: 0

```

Listing 254: Ausgabe Testfall 5 Prompt 1 Ausführung 2

```

1 Traceback (most recent call last):
2   File "/home/mauke/Documents/Uni/Bachelorarbeit/venv/lib/python3.13/site-
   packages/pandas/core/indexes/base.py", line 3805, in get_loc
3     return self._engine.get_loc(casted_key)
4         ~~~~~
5   File "index.pyx", line 167, in pandas._libs.index.IndexEngine.get_loc
6   File "index.pyx", line 196, in pandas._libs.index.IndexEngine.get_loc
7   File "pandas/_libs/hashtable_class_helper.pxi", line 7081, in pandas._libs.
   hashtable.PyObjectHashTable.get_item
8   File "pandas/_libs/hashtable_class_helper.pxi", line 7089, in pandas._libs.
   hashtable.PyObjectHashTable.get_item
9   KeyError: 'LOR-Schlüssel'
10
11 The above exception was the direct cause of the following exception:
12
13 Traceback (most recent call last):
14   File "/home/mauke/Documents/Uni/Bachelorarbeit/testcases/testcase5/prompt1/
   exec3/script.py", line 15, in <module>
15     df['Oberbezirk_Key'] = df['LOR-Schlüssel'].astype(str).str[:2]
16         ~~~~~
17   File "/home/mauke/Documents/Uni/Bachelorarbeit/venv/lib/python3.13/site-
   packages/pandas/core/frame.py", line 4102, in __getitem__
18     indexer = self.columns.get_loc(key)
19   File "/home/mauke/Documents/Uni/Bachelorarbeit/venv/lib/python3.13/site-
   packages/pandas/core/indexes/base.py", line 3812, in get_loc
20     raise KeyError(key) from err
21   KeyError: 'LOR-Schlüssel'
22 Command exited with non-zero status 1

```

```

23 Command being timed: "python testcase5/prompt1/exec3/script.py"
24 User time (seconds): 2.40
25 System time (seconds): 0.08
26 Percent of CPU this job got: 436%
27 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.56
28 Average shared text size (kbytes): 0
29 Average unshared data size (kbytes): 0
30 Average stack size (kbytes): 0
31 Average total size (kbytes): 0
32 Maximum resident set size (kbytes): 156760
33 Average resident set size (kbytes): 0
34 Major (requiring I/O) page faults: 0
35 Minor (reclaiming a frame) page faults: 33052
36 Voluntary context switches: 34
37 Involuntary context switches: 253
38 Swaps: 0
39 File system inputs: 0
40 File system outputs: 0
41 Socket messages sent: 0
42 Socket messages received: 0
43 Signals delivered: 0
44 Page size (bytes): 4096
45 Exit status: 1

```

Listing 255: Ausgabe Testfall 5Prompt 1Ausführung 3

```

1  Traceback (most recent call last):
2    File "/home/mauke/Documents/Uni/Bachelorarbeit/venv/lib/python3.13/site-
   packages/pandas/core/indexes/base.py", line 3805, in get_loc
3     return self._engine.get_loc(casted_key)
4     ~~~~~
5    File "index.pyx", line 167, in pandas._libs.index.IndexEngine.get_loc
6    File "index.pyx", line 196, in pandas._libs.index.IndexEngine.get_loc
7    File "pandas/_libs/hashtable_class_helper.pxi", line 7081, in pandas._libs.
   hashtable.PyObjectHashTable.get_item
8    File "pandas/_libs/hashtable_class_helper.pxi", line 7089, in pandas._libs.
   hashtable.PyObjectHashTable.get_item
9  KeyError: 'LOR-Schlüssel'
10
11 The above exception was the direct cause of the following exception:
12
13 Traceback (most recent call last):
14    File "/home/mauke/Documents/Uni/Bachelorarbeit/testcases/testcase5/prompt1/
   exec4/script.py", line 70, in <module>
15     finde_oberbezirke_mit_meisten_raub(excel_datei, sheet_name)
16     ~~~~~
17    File "/home/mauke/Documents/Uni/Bachelorarbeit/testcases/testcase5/prompt1/
   exec4/script.py", line 12, in finde_oberbezirke_mit_meisten_raub
18     df['LOR-Schlüssel'] = df['LOR-Schlüssel'].str.strip().str.zfill(6)
19     ~~~~~
20    File "/home/mauke/Documents/Uni/Bachelorarbeit/venv/lib/python3.13/site-
   packages/pandas/core/frame.py", line 4102, in __getitem__
21     indexer = self.columns.get_loc(key)

```

```

22 File "/home/mauke/Documents/Uni/Bachelorarbeit/venv/lib/python3.13/site-
    packages/pandas/core/indexes/base.py", line 3812, in get_loc
23     raise KeyError(key) from err
24 KeyError: 'LOR-Schlüssel'
25 Command exited with non-zero status 1
26 Command being timed: "python testcase5/prompt1/exec4/script.py"
27 User time (seconds): 2.43
28 System time (seconds): 0.06
29 Percent of CPU this job got: 451%
30 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.55
31 Average shared text size (kbytes): 0
32 Average unshared data size (kbytes): 0
33 Average stack size (kbytes): 0
34 Average total size (kbytes): 0
35 Maximum resident set size (kbytes): 155968
36 Average resident set size (kbytes): 0
37 Major (requiring I/O) page faults: 0
38 Minor (reclaiming a frame) page faults: 33144
39 Voluntary context switches: 36
40 Involuntary context switches: 453
41 Swaps: 0
42 File system inputs: 0
43 File system outputs: 0
44 Socket messages sent: 0
45 Socket messages received: 0
46 Signals delivered: 0
47 Page size (bytes): 4096
48 Exit status: 1

```

Listing 256: Ausgabe Testfall 5Prompt 1Ausführung 4

```

1 Traceback (most recent call last):
2   File "/home/mauke/Documents/Uni/Bachelorarbeit/venv/lib/python3.13/site-
    packages/pandas/core/indexes/base.py", line 3805, in get_loc
3     return self._engine.get_loc(casted_key)
4     ~~~~~
5   File "index.pyx", line 167, in pandas._libs.index.IndexEngine.get_loc
6   File "index.pyx", line 196, in pandas._libs.index.IndexEngine.get_loc
7   File "pandas/_libs/hashtable_class_helper.pxi", line 7081, in pandas._libs.
    hashtable.PyObjectHashTable.get_item
8   File "pandas/_libs/hashtable_class_helper.pxi", line 7089, in pandas._libs.
    hashtable.PyObjectHashTable.get_item
9 KeyError: 'LOR-Schlüssel'
10
11 The above exception was the direct cause of the following exception:
12
13 Traceback (most recent call last):
14   File "/home/mauke/Documents/Uni/Bachelorarbeit/testcases/testcase5/prompt1/
    exec5/script.py", line 13, in <module>
15     df['LOR-Schlüssel'] = df['LOR-Schlüssel'].astype(str)
16     ~~~~~
17   File "/home/mauke/Documents/Uni/Bachelorarbeit/venv/lib/python3.13/site-
    packages/pandas/core/frame.py", line 4102, in __getitem__

```

```

18     indexer = self.columns.get_loc(key)
19     File "/home/mauke/Documents/Uni/Bachelorarbeit/venv/lib/python3.13/site-
    packages/pandas/core/indexes/base.py", line 3812, in get_loc
20         raise KeyError(key) from err
21 KeyError: 'LOR-Schlüssel'
22 Command exited with non-zero status 1
23     Command being timed: "python testcase5/prompt1/exec5/script.py"
24     User time (seconds): 2.43
25     System time (seconds): 0.07
26     Percent of CPU this job got: 452%
27     Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.55
28     Average shared text size (kbytes): 0
29     Average unshared data size (kbytes): 0
30     Average stack size (kbytes): 0
31     Average total size (kbytes): 0
32     Maximum resident set size (kbytes): 156216
33     Average resident set size (kbytes): 0
34     Major (requiring I/O) page faults: 0
35     Minor (reclaiming a frame) page faults: 33066
36     Voluntary context switches: 33
37     Involuntary context switches: 416
38     Swaps: 0
39     File system inputs: 0
40     File system outputs: 0
41     Socket messages sent: 0
42     Socket messages received: 0
43     Signals delivered: 0
44     Page size (bytes): 4096
45     Exit status: 1

```

Listing 257: Ausgabe Testfall 5Prompt 1Ausführung 5

```

1 Unterbezirk mit den meisten Raubdelikten pro Oberbezirk:
2 Mitte: Alexanderplatz (199 Raubdelikte)
3 Friedrichshain-Kreuzberg: Südliche Luisenstadt (256 Raubdelikte)
4 Pankow: Prenzlauer Berg Ost (70 Raubdelikte)
5 Charlottenburg-Wilmersdorf: Lietzenburger Straße (116 Raubdelikte)
6 Spandau: Spandau Mitte (61 Raubdelikte)
7 Steglitz-Zehlendorf: Schloßstraße (43 Raubdelikte)
8 Tempelhof-Schöneberg: Schöneberg Nordwest (107 Raubdelikte)
9 Neukölln: Reuterstraße (92 Raubdelikte)
10 Treptow-Köpenick: Alt-Treptow (31 Raubdelikte)
11 Marzahn-Hellersdorf: Marzahn Mitte (70 Raubdelikte)
12 Lichtenberg: Neu-Hohenschönhausen Süd (53 Raubdelikte)
13 Reinickendorf: Ost 2 - Alt-Reinickendorf (56 Raubdelikte)
14     Command being timed: "python testcase5/prompt2/exec1/script.py"
15     User time (seconds): 2.41
16     System time (seconds): 0.06
17     Percent of CPU this job got: 464%
18     Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.53
19     Average shared text size (kbytes): 0
20     Average unshared data size (kbytes): 0
21     Average stack size (kbytes): 0

```

```

22 Average total size (kbytes): 0
23 Maximum resident set size (kbytes): 155696
24 Average resident set size (kbytes): 0
25 Major (requiring I/O) page faults: 0
26 Minor (reclaiming a frame) page faults: 33186
27 Voluntary context switches: 34
28 Involuntary context switches: 552
29 Swaps: 0
30 File system inputs: 0
31 File system outputs: 0
32 Socket messages sent: 0
33 Socket messages received: 0
34 Signals delivered: 0
35 Page size (bytes): 4096
36 Exit status: 0

```

Listing 258: Ausgabe Testfall 5Prompt 2Ausführung 1

```

1 Unterbezirk mit den meisten Raubdelikten pro Oberbezirk (2023):
2
3 Mitte: Alexanderplatz (199 Raubdelikte)
4 Friedrichshain-Kreuzberg: Südliche Luisenstadt (256 Raubdelikte)
5 Pankow: Prenzlauer Berg Ost (70 Raubdelikte)
6 Charlottenburg-Wilmersdorf: Lietzenburger Straße (116 Raubdelikte)
7 Spandau: Spandau Mitte (61 Raubdelikte)
8 Steglitz-Zehlendorf: Schloßstraße (43 Raubdelikte)
9 Tempelhof-Schöneberg: Schöneberg Nordwest (107 Raubdelikte)
10 Neukölln: Reuterstraße (92 Raubdelikte)
11 Treptow-Köpenick: Alt-Treptow (31 Raubdelikte)
12 Marzahn-Hellersdorf: Marzahn Mitte (70 Raubdelikte)
13 Lichtenberg: Neu-Hohenschönhausen Süd (53 Raubdelikte)
14 Reinickendorf: Ost 2 - Alt-Reinickendorf (56 Raubdelikte)
15 Command being timed: "python testcase5/prompt2/exec2/script.py"
16 User time (seconds): 2.43
17 System time (seconds): 0.06
18 Percent of CPU this job got: 457%
19 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.54
20 Average shared text size (kbytes): 0
21 Average unshared data size (kbytes): 0
22 Average stack size (kbytes): 0
23 Average total size (kbytes): 0
24 Maximum resident set size (kbytes): 155644
25 Average resident set size (kbytes): 0
26 Major (requiring I/O) page faults: 0
27 Minor (reclaiming a frame) page faults: 33056
28 Voluntary context switches: 37
29 Involuntary context switches: 470
30 Swaps: 0
31 File system inputs: 0
32 File system outputs: 0
33 Socket messages sent: 0
34 Socket messages received: 0
35 Signals delivered: 0

```

```
36 Page size (bytes): 4096
37 Exit status: 0
```

Listing 259: Ausgabe Testfall 5Prompt 2Ausführung 2

```
1 Unterbezirk mit den meisten Raubdelikten pro Oberbezirk (2023):
2
3 Oberbezirk: Charlottenburg-Wilmersdorf
4   Unterbezirk: Lietzenburger Straße
5   Anzahl Raubdelikte: 116
6
7 Oberbezirk: Friedrichshain-Kreuzberg
8   Unterbezirk: Südliche Luisenstadt
9   Anzahl Raubdelikte: 256
10
11 Oberbezirk: Lichtenberg
12   Unterbezirk: Neu-Hohenschönhausen Süd
13   Anzahl Raubdelikte: 53
14
15 Oberbezirk: Marzahn-Hellersdorf
16   Unterbezirk: Marzahn Mitte
17   Anzahl Raubdelikte: 70
18
19 Oberbezirk: Mitte
20   Unterbezirk: Alexanderplatz
21   Anzahl Raubdelikte: 199
22
23 Oberbezirk: Neukölln
24   Unterbezirk: Reuterstraße
25   Anzahl Raubdelikte: 92
26
27 Oberbezirk: Pankow
28   Unterbezirk: Prenzlauer Berg Ost
29   Anzahl Raubdelikte: 70
30
31 Oberbezirk: Reinickendorf
32   Unterbezirk: Ost 2 - Alt-Reinickendorf
33   Anzahl Raubdelikte: 56
34
35 Oberbezirk: Spandau
36   Unterbezirk: Spandau Mitte
37   Anzahl Raubdelikte: 61
38
39 Oberbezirk: Steglitz-Zehlendorf
40   Unterbezirk: Schloßstraße
41   Anzahl Raubdelikte: 43
42
43 Oberbezirk: Tempelhof-Schöneberg
44   Unterbezirk: Schöneberg Nordwest
45   Anzahl Raubdelikte: 107
46
47 Oberbezirk: Treptow-Köpenick
48   Unterbezirk: Alt-Treptow
```

```

49 Anzahl Raubdelikte: 31
50
51 Command being timed: "python testcase5/prompt2/exec3/script.py"
52 User time (seconds): 2.37
53 System time (seconds): 0.05
54 Percent of CPU this job got: 423%
55 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.57
56 Average shared text size (kbytes): 0
57 Average unshared data size (kbytes): 0
58 Average stack size (kbytes): 0
59 Average total size (kbytes): 0
60 Maximum resident set size (kbytes): 156852
61 Average resident set size (kbytes): 0
62 Major (requiring I/O) page faults: 0
63 Minor (reclaiming a frame) page faults: 33130
64 Voluntary context switches: 41
65 Involuntary context switches: 248
66 Swaps: 0
67 File system inputs: 0
68 File system outputs: 0
69 Socket messages sent: 0
70 Socket messages received: 0
71 Signals delivered: 0
72 Page size (bytes): 4096
73 Exit status: 0

```

Listing 260: Ausgabe Testfall 5Prompt 2Ausführung 3

```

1 Unterbezirke mit den meisten Raubdelikten pro Oberbezirk:
2
3 Oberbezirk: Mitte
4   Unterbezirk: Alexanderplatz
5   Raubdelikte: 199
6
7 Oberbezirk: Friedrichshain-Kreuzberg
8   Unterbezirk: Südliche Luisenstadt
9   Raubdelikte: 256
10
11 Oberbezirk: Pankow
12   Unterbezirk: Prenzlauer Berg Ost
13   Raubdelikte: 70
14
15 Oberbezirk: Charlottenburg-Wilmersdorf
16   Unterbezirk: Lietzenburger Straße
17   Raubdelikte: 116
18
19 Oberbezirk: Spandau
20   Unterbezirk: Spandau Mitte
21   Raubdelikte: 61
22
23 Oberbezirk: Steglitz-Zehlendorf
24   Unterbezirk: Schloßstraße
25   Raubdelikte: 43

```



```

26
27 Oberbezirk: Tempelhof-Schöneberg
28   Unterbezirk: Schöneberg Nordwest
29   Raubdelikte: 107
30
31 Oberbezirk: Neukölln
32   Unterbezirk: Reuterstraße
33   Raubdelikte: 92
34
35 Oberbezirk: Treptow-Köpenick
36   Unterbezirk: Alt-Treptow
37   Raubdelikte: 31
38
39 Oberbezirk: Marzahn-Hellersdorf
40   Unterbezirk: Marzahn Mitte
41   Raubdelikte: 70
42
43 Oberbezirk: Lichtenberg
44   Unterbezirk: Neu-Hohenschönhausen Süd
45   Raubdelikte: 53
46
47 Oberbezirk: Reinickendorf
48   Unterbezirk: Ost 2 - Alt-Reinickendorf
49   Raubdelikte: 56
50
51 Command being timed: "python testcase5/prompt2/exec4/script.py"
52 User time (seconds): 2.40
53 System time (seconds): 0.06
54 Percent of CPU this job got: 440%
55 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.56
56 Average shared text size (kbytes): 0
57 Average unshared data size (kbytes): 0
58 Average stack size (kbytes): 0
59 Average total size (kbytes): 0
60 Maximum resident set size (kbytes): 156876
61 Average resident set size (kbytes): 0
62 Major (requiring I/O) page faults: 0
63 Minor (reclaiming a frame) page faults: 33026
64 Voluntary context switches: 43
65 Involuntary context switches: 328
66 Swaps: 0
67 File system inputs: 0
68 File system outputs: 0
69 Socket messages sent: 0
70 Socket messages received: 0
71 Signals delivered: 0
72 Page size (bytes): 4096
73 Exit status: 0

```

Listing 261: Ausgabe Testfall 5Prompt 2Ausführung 4

```

1 Oberbezirk: Mitte
2   Unterbezirk mit den meisten Raubdelikten: Alexanderplatz (199 Raubdelikte)

```

```

3
4 Oberbezirk: Friedrichshain-Kreuzberg
5   Unterbezirk mit den meisten Raubdelikten: Südliche Luisenstadt (256
6     Raubdelikte)
7
8 Oberbezirk: Pankow
9   Unterbezirk mit den meisten Raubdelikten: Prenzlauer Berg Ost (70
10     Raubdelikte)
11
12 Oberbezirk: Charlottenburg-Wilmersdorf
13   Unterbezirk mit den meisten Raubdelikten: Lietzenburger Straße (116
14     Raubdelikte)
15
16 Oberbezirk: Spandau
17   Unterbezirk mit den meisten Raubdelikten: Spandau Mitte (61 Raubdelikte)
18
19 Oberbezirk: Steglitz-Zehlendorf
20   Unterbezirk mit den meisten Raubdelikten: Schloßstraße (43 Raubdelikte)
21
22 Oberbezirk: Tempelhof-Schöneberg
23   Unterbezirk mit den meisten Raubdelikten: Schöneberg Nordwest (107
24     Raubdelikte)
25
26 Oberbezirk: Neukölln
27   Unterbezirk mit den meisten Raubdelikten: Reuterstraße (92 Raubdelikte)
28
29 Oberbezirk: Treptow-Köpenick
30   Unterbezirk mit den meisten Raubdelikten: Alt-Treptow (31 Raubdelikte)
31
32 Oberbezirk: Marzahn-Hellersdorf
33   Unterbezirk mit den meisten Raubdelikten: Marzahn Mitte (70 Raubdelikte)
34
35 Oberbezirk: Lichtenberg
36   Unterbezirk mit den meisten Raubdelikten: Neu-Hohenschönhausen Süd (53
37     Raubdelikte)
38
39 Oberbezirk: Reinickendorf
40   Unterbezirk mit den meisten Raubdelikten: Ost 2 - Alt-Reinickendorf (56
41     Raubdelikte)
42
43 Command being timed: "python testcase5/prompt2/exec5/script.py"
44 User time (seconds): 2.41
45 System time (seconds): 0.05
46 Percent of CPU this job got: 440%
47 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.56
48 Average shared text size (kbytes): 0
49 Average unshared data size (kbytes): 0
50 Average stack size (kbytes): 0
51 Average total size (kbytes): 0
52 Maximum resident set size (kbytes): 156640
53 Average resident set size (kbytes): 0
54 Major (requiring I/O) page faults: 0

```

```

49 Minor (reclaiming a frame) page faults: 33139
50 Voluntary context switches: 33
51 Involuntary context switches: 332
52 Swaps: 0
53 File system inputs: 0
54 File system outputs: 0
55 Socket messages sent: 0
56 Socket messages received: 0
57 Signals delivered: 0
58 Page size (bytes): 4096
59 Exit status: 0

```

Listing 262: Ausgabe Testfall 5Prompt 2Ausführung 5

```

1
2 0 Oberbezirk Unterbezirk Raub
3 1 Charlottenburg-Wilmersdorf Lietzenburger Straße 116
4 2 Friedrichshain-Kreuzberg Südliche Luisenstadt 256
5 3 Lichtenberg Neu-Hohenschönhausen Süd 53
6 4 Marzahn-Hellersdorf Marzahn Mitte 70
7 5 Mitte Alexanderplatz 199
8 6 Neukölln Reuterstraße 92
9 7 Pankow Prenzlauer Berg Ost 70
10 8 Reinickendorf Ost 2 - Alt-Reinickendorf 56
11 9 Spandau Spandau Mitte 61
12 10 Steglitz-Zehlendorf Schloßstraße 43
13 11 Tempelhof-Schöneberg Schöneberg Nordwest 107
14 12 Treptow-Köpenick Alt-Treptow 31
15 Command being timed: "python testcase5/prompt3/exec1/script.py"
16 User time (seconds): 2.41
17 System time (seconds): 0.06
18 Percent of CPU this job got: 417%
19 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.59
20 Average shared text size (kbytes): 0
21 Average unshared data size (kbytes): 0
22 Average stack size (kbytes): 0
23 Average total size (kbytes): 0
24 Maximum resident set size (kbytes): 156332
25 Average resident set size (kbytes): 0
26 Major (requiring I/O) page faults: 0
27 Minor (reclaiming a frame) page faults: 33186
28 Voluntary context switches: 45
29 Involuntary context switches: 606
30 Swaps: 0
31 File system inputs: 0
32 File system outputs: 0
33 Socket messages sent: 0
34 Socket messages received: 0
35 Signals delivered: 0
36 Page size (bytes): 4096
Exit status: 0

```

Listing 263: Ausgabe Testfall 5Prompt 3Ausführung 1

```

1      Oberbezirk      Unterbezirk      Raub
2 0  Charlottenburg-Wilmersdorf      Lietzenburger Straße      116
3 1  Friedrichshain-Kreuzberg      Südliche Luisenstadt      256
4 2      Lichtenberg      Neu-Hohenschönhausen Süd      53
5 3      Marzahn-Hellersdorf      Marzahn Mitte      70
6 4      Mitte      Alexanderplatz      199
7 5      Neukölln      Reuterstraße      92
8 6      Pankow      Prenzlauer Berg Ost      70
9 7      Reinickendorf      Ost 2 - Alt-Reinickendorf      56
10 8      Spandau      Spandau Mitte      61
11 9      Steglitz-Zehlendorf      Schloßstraße      43
12 10      Tempelhof-Schöneberg      Schöneberg Northwest      107
13 11      Treptow-Köpenick      Alt-Treptow      31
14 Command being timed: "python testcase5/prompt3/exec2/script.py"
15 User time (seconds): 2.41
16 System time (seconds): 0.06
17 Percent of CPU this job got: 430%
18 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.57
19 Average shared text size (kbytes): 0
20 Average unshared data size (kbytes): 0
21 Average stack size (kbytes): 0
22 Average total size (kbytes): 0
23 Maximum resident set size (kbytes): 156104
24 Average resident set size (kbytes): 0
25 Major (requiring I/O) page faults: 0
26 Minor (reclaiming a frame) page faults: 33016
27 Voluntary context switches: 34
28 Involuntary context switches: 280
29 Swaps: 0
30 File system inputs: 0
31 File system outputs: 0
32 Socket messages sent: 0
33 Socket messages received: 0
34 Signals delivered: 0
35 Page size (bytes): 4096
36 Exit status: 0

```

Listing 264: Ausgabe Testfall 5Prompt 3Ausführung 2

```

1      Oberbezirk      Unterbezirk      Raub
2 0  Charlottenburg-Wilmersdorf      Lietzenburger Straße      116
3 1  Friedrichshain-Kreuzberg      Südliche Luisenstadt      256
4 2      Lichtenberg      Neu-Hohenschönhausen Süd      53
5 3      Marzahn-Hellersdorf      Marzahn Mitte      70
6 4      Mitte      Alexanderplatz      199
7 5      Neukölln      Reuterstraße      92
8 6      Pankow      Prenzlauer Berg Ost      70
9 7      Reinickendorf      Ost 2 - Alt-Reinickendorf      56
10 8      Spandau      Spandau Mitte      61
11 9      Steglitz-Zehlendorf      Schloßstraße      43
12 10      Tempelhof-Schöneberg      Schöneberg Northwest      107
13 11      Treptow-Köpenick      Alt-Treptow      31

```

```

14 Command being timed: "python testcase5/prompt3/exec3/script.py"
15 User time (seconds): 2.44
16 System time (seconds): 0.06
17 Percent of CPU this job got: 450%
18 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.55
19 Average shared text size (kbytes): 0
20 Average unshared data size (kbytes): 0
21 Average stack size (kbytes): 0
22 Average total size (kbytes): 0
23 Maximum resident set size (kbytes): 156068
24 Average resident set size (kbytes): 0
25 Major (requiring I/O) page faults: 0
26 Minor (reclaiming a frame) page faults: 33033
27 Voluntary context switches: 44
28 Involuntary context switches: 381
29 Swaps: 0
30 File system inputs: 0
31 File system outputs: 0
32 Socket messages sent: 0
33 Socket messages received: 0
34 Signals delivered: 0
35 Page size (bytes): 4096
36 Exit status: 0

```

Listing 265: Ausgabe Testfall 5Prompt 3Ausführung 3

```

1 Sheet 'Fallzahlen_2023' erfolgreich geladen.
2 DataFrame auf notwendige Spalten reduziert.
3 Letzte zwei Zeilen (Gesamtwerte) entfernt.
4 Oberbezirke zugewiesen.
5 Unterbezirke von Oberbezirken getrennt.
6
7 Unterbezirke mit den höchsten 'Raub'-Werten je Oberbezirk:
8
9      Oberbezirk      Unterbezirk  Raub
10 0 Charlottenburg-Wilmersdorf  Lietzenburger Straße  116
11 1 Friedrichshain-Kreuzberg    Südliche Luisenstadt  256
12 2 Lichtenberg      Neu-Hohenschönhausen Süd    53
13 3 Marzahn-Hellersdorf      Marzahn Mitte    70
14 4 Mitte      Alexanderplatz  199
15 5 Neukölln      Reuterstraße    92
16 6 Pankow      Prenzlauer Berg Ost    70
17 7 Reinickendorf Ost 2 - Alt-Reinickendorf  56
18 8 Spandau      Spandau Mitte    61
19 9 Steglitz-Zehlendorf      Schloßstraße    43
20 10 Tempelhof-Schöneberg      Schöneberg Nordwest  107
21 11 Treptow-Köpenick      Alt-Treptow    31
22
23 Ergebnisse wurden in 'Max_Raub_Unterbezirke_2023.xlsx' gespeichert.
24 Command being timed: "python testcase5/prompt3/exec4/script.py"
25 User time (seconds): 2.42
26 System time (seconds): 0.04
27 Percent of CPU this job got: 424%
28 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.58

```

```

28 Average shared text size (kbytes): 0
29 Average unshared data size (kbytes): 0
30 Average stack size (kbytes): 0
31 Average total size (kbytes): 0
32 Maximum resident set size (kbytes): 156884
33 Average resident set size (kbytes): 0
34 Major (requiring I/O) page faults: 0
35 Minor (reclaiming a frame) page faults: 33151
36 Voluntary context switches: 35
37 Involuntary context switches: 434
38 Swaps: 0
39 File system inputs: 0
40 File system outputs: 176
41 Socket messages sent: 0
42 Socket messages received: 0
43 Signals delivered: 0
44 Page size (bytes): 4096
45 Exit status: 0

```

Listing 266: Ausgabe Testfall 5Prompt 3Ausführung 4

```

1 Oberbezirk Unterbezirk Raub
2 0 Charlottenburg-Wilmersdorf Lietzenburger Straße 116
3 1 Friedrichshain-Kreuzberg Südliche Luisenstadt 256
4 2 Lichtenberg Neu-Hohenschönhausen Süd 53
5 3 Marzahn-Hellersdorf Marzahn Mitte 70
6 4 Mitte Alexanderplatz 199
7 5 Neukölln Reuterstraße 92
8 6 Pankow Prenzlauer Berg Ost 70
9 7 Reinickendorf Ost 2 - Alt-Reinickendorf 56
10 8 Spandau Spandau Mitte 61
11 9 Steglitz-Zehlendorf Schloßstraße 43
12 10 Tempelhof-Schöneberg Schöneberg Nordwest 107
13 11 Treptow-Köpenick Alt-Treptow 31
14 Command being timed: "python testcase5/prompt3/exec5/script.py"
15 User time (seconds): 2.36
16 System time (seconds): 0.09
17 Percent of CPU this job got: 418%
18 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.58
19 Average shared text size (kbytes): 0
20 Average unshared data size (kbytes): 0
21 Average stack size (kbytes): 0
22 Average total size (kbytes): 0
23 Maximum resident set size (kbytes): 156744
24 Average resident set size (kbytes): 0
25 Major (requiring I/O) page faults: 0
26 Minor (reclaiming a frame) page faults: 33022
27 Voluntary context switches: 39
28 Involuntary context switches: 228
29 Swaps: 0
30 File system inputs: 0
31 File system outputs: 0
32 Socket messages sent: 0

```

```

33 Socket messages received: 0
34 Signals delivered: 0
35 Page size (bytes): 4096
36 Exit status: 0

```

Listing 267: Ausgabe Testfall 5 Prompt 3 Ausführung 5

11.3.19 Komplette Ausgaben der Skripte Testfall 6

```

1 Wichtige Spalten fehlen im Sheet Fallzahlen_2014.
2 Wichtige Spalten fehlen im Sheet Fallzahlen_2015.
3 Wichtige Spalten fehlen im Sheet Fallzahlen_2016.
4 Wichtige Spalten fehlen im Sheet Fallzahlen_2017.
5 Wichtige Spalten fehlen im Sheet Fallzahlen_2018.
6 Wichtige Spalten fehlen im Sheet Fallzahlen_2019.
7 Wichtige Spalten fehlen im Sheet Fallzahlen_2020.
8 Wichtige Spalten fehlen im Sheet Fallzahlen_2021.
9 Wichtige Spalten fehlen im Sheet Fallzahlen_2022.
10 Wichtige Spalten fehlen im Sheet Fallzahlen_2023.
11 Keine Daten zum Verarbeiten gefunden.
12 Command being timed: "python testcase6/prompt1/exec1/script.py"
13 User time (seconds): 5.24
14 System time (seconds): 0.25
15 Percent of CPU this job got: 152%
16 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:03.60
17 Average shared text size (kbytes): 0
18 Average unshared data size (kbytes): 0
19 Average stack size (kbytes): 0
20 Average total size (kbytes): 0
21 Maximum resident set size (kbytes): 157408
22 Average resident set size (kbytes): 0
23 Major (requiring I/O) page faults: 0
24 Minor (reclaiming a frame) page faults: 171208
25 Voluntary context switches: 33
26 Involuntary context switches: 289
27 Swaps: 0
28 File system inputs: 0
29 File system outputs: 0
30 Socket messages sent: 0
31 Socket messages received: 0
32 Signals delivered: 0
33 Page size (bytes): 4096
34 Exit status: 0

```

Listing 268: Ausgabe Testfall 6 Prompt 1 Ausführung 1

```

1 Wichtige Spalten fehlen im Sheet Fallzahlen_2014. Überspringe dieses Sheet.
2 Wichtige Spalten fehlen im Sheet Fallzahlen_2015. Überspringe dieses Sheet.
3 Wichtige Spalten fehlen im Sheet Fallzahlen_2016. Überspringe dieses Sheet.
4 Wichtige Spalten fehlen im Sheet Fallzahlen_2017. Überspringe dieses Sheet.
5 Wichtige Spalten fehlen im Sheet Fallzahlen_2018. Überspringe dieses Sheet.
6 Wichtige Spalten fehlen im Sheet Fallzahlen_2019. Überspringe dieses Sheet.

```

```

7 Wichtige Spalten fehlen im Sheet Fallzahlen_2020. Überspringe dieses Sheet.
8 Wichtige Spalten fehlen im Sheet Fallzahlen_2021. Überspringe dieses Sheet.
9 Wichtige Spalten fehlen im Sheet Fallzahlen_2022. Überspringe dieses Sheet.
10 Wichtige Spalten fehlen im Sheet Fallzahlen_2023. Überspringe dieses Sheet.
11 Traceback (most recent call last):
12   File "/home/mauke/Documents/Uni/Bachelorarbeit/testcases/testcase6/prompt1/
      exec2/script.py", line 52, in <module>
13     top_10_unterbezirke = get_top_10_unterbezirke(excel_datei)
14   File "/home/mauke/Documents/Uni/Bachelorarbeit/testcases/testcase6/prompt1/
      exec2/script.py", line 43, in get_top_10_unterbezirke
15     aggregierte_daten_sorted = aggregierte_daten.sort_values(by='Straftaten
      insgesamt', ascending=False)
16   File "/home/mauke/Documents/Uni/Bachelorarbeit/venv/lib/python3.13/site-
      packages/pandas/core/frame.py", line 7189, in sort_values
17     k = self._get_label_or_level_values(by[0], axis=axis)
18   File "/home/mauke/Documents/Uni/Bachelorarbeit/venv/lib/python3.13/site-
      packages/pandas/core/generic.py", line 1911, in _get_label_or_level_values
19     raise KeyError(key)
20 KeyError: 'Straftaten insgesamt'
21 Command exited with non-zero status 1
22 Command being timed: "python testcase6/prompt1/exec2/script.py"
23 User time (seconds): 5.21
24 System time (seconds): 0.25
25 Percent of CPU this job got: 151%
26 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:03.61
27 Average shared text size (kbytes): 0
28 Average unshared data size (kbytes): 0
29 Average stack size (kbytes): 0
30 Average total size (kbytes): 0
31 Maximum resident set size (kbytes): 157956
32 Average resident set size (kbytes): 0
33 Major (requiring I/O) page faults: 0
34 Minor (reclaiming a frame) page faults: 171520
35 Voluntary context switches: 36
36 Involuntary context switches: 508
37 Swaps: 0
38 File system inputs: 0
39 File system outputs: 0
40 Socket messages sent: 0
41 Socket messages received: 0
42 Signals delivered: 0
43 Page size (bytes): 4096
44 Exit status: 1

```

Listing 269: Ausgabe Testfall 6Prompt 1Ausführung 2

```

1 Die Top 10 Unterbezirke mit den meisten Straftaten (2014-2023):
2 Bezeichnung (Bezirksregion)
3 Berlin (PKS gesamt)                    5269896
4 Mitte                                878048
5 Friedrichshain-Kreuzberg              579591
6 Charlottenburg-Wilmersdorf           519706
7 Stadtgebiet Berlin, nicht zuzuordnen  512158

```



```

8 Neukölln 436568
9 Tempelhof-Schöneberg 398767
10 Pankow 393574
11 Reinickendorf 284193
12 Lichtenberg 279018
13 Name: Straftaten \n-insgesamt-, dtype: int64
14 Command being timed: "python testcase6/prompt1/exec3/script.py"
15 User time (seconds): 5.34
16 System time (seconds): 0.24
17 Percent of CPU this job got: 153%
18 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:03.65
19 Average shared text size (kbytes): 0
20 Average unshared data size (kbytes): 0
21 Average stack size (kbytes): 0
22 Average total size (kbytes): 0
23 Maximum resident set size (kbytes): 157480
24 Average resident set size (kbytes): 0
25 Major (requiring I/O) page faults: 0
26 Minor (reclaiming a frame) page faults: 171785
27 Voluntary context switches: 38
28 Involuntary context switches: 271
29 Swaps: 0
30 File system inputs: 0
31 File system outputs: 0
32 Socket messages sent: 0
33 Socket messages received: 0
34 Signals delivered: 0
35 Page size (bytes): 4096
36 Exit status: 0

```

Listing 270: Ausgabe Testfall 6Prompt 1Ausführung 3

```

1 Warnung: Nicht alle notwendigen Spalten in Sheet Fallzahlen_2014 gefunden.
2 Warnung: Nicht alle notwendigen Spalten in Sheet Fallzahlen_2015 gefunden.
3 Warnung: Nicht alle notwendigen Spalten in Sheet Fallzahlen_2016 gefunden.
4 Warnung: Nicht alle notwendigen Spalten in Sheet Fallzahlen_2017 gefunden.
5 Warnung: Nicht alle notwendigen Spalten in Sheet Fallzahlen_2018 gefunden.
6 Warnung: Nicht alle notwendigen Spalten in Sheet Fallzahlen_2019 gefunden.
7 Warnung: Nicht alle notwendigen Spalten in Sheet Fallzahlen_2020 gefunden.
8 Warnung: Nicht alle notwendigen Spalten in Sheet Fallzahlen_2021 gefunden.
9 Warnung: Nicht alle notwendigen Spalten in Sheet Fallzahlen_2022 gefunden.
10 Warnung: Nicht alle notwendigen Spalten in Sheet Fallzahlen_2023 gefunden.
11 Top 10 Unterbezirke mit den meisten Straftaten (2014-2023):
12 Empty DataFrame
13 Columns: [Bezirksregion, Gesamt_Straftaten]
14 Index: []
15 Command being timed: "python testcase6/prompt1/exec4/script.py"
16 User time (seconds): 5.14
17 System time (seconds): 0.29
18 Percent of CPU this job got: 150%
19 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:03.60
20 Average shared text size (kbytes): 0
21 Average unshared data size (kbytes): 0

```

```

22 Average stack size (kbytes): 0
23 Average total size (kbytes): 0
24 Maximum resident set size (kbytes): 158896
25 Average resident set size (kbytes): 0
26 Major (requiring I/O) page faults: 0
27 Minor (reclaiming a frame) page faults: 173547
28 Voluntary context switches: 38
29 Involuntary context switches: 1004
30 Swaps: 0
31 File system inputs: 0
32 File system outputs: 0
33 Socket messages sent: 0
34 Socket messages received: 0
35 Signals delivered: 0
36 Page size (bytes): 4096
37 Exit status: 0

```

Listing 271: Ausgabe Testfall 6Prompt 1Ausführung 4

```

1 Wichtige Spalten fehlen im Sheet Fallzahlen_2014.
2 Wichtige Spalten fehlen im Sheet Fallzahlen_2015.
3 Wichtige Spalten fehlen im Sheet Fallzahlen_2016.
4 Wichtige Spalten fehlen im Sheet Fallzahlen_2017.
5 Wichtige Spalten fehlen im Sheet Fallzahlen_2018.
6 Wichtige Spalten fehlen im Sheet Fallzahlen_2019.
7 Wichtige Spalten fehlen im Sheet Fallzahlen_2020.
8 Wichtige Spalten fehlen im Sheet Fallzahlen_2021.
9 Wichtige Spalten fehlen im Sheet Fallzahlen_2022.
10 Wichtige Spalten fehlen im Sheet Fallzahlen_2023.
11 Keine Daten zum Verarbeiten gefunden.
12 Command being timed: "python testcase6/prompt1/exec1/script.py"
13 User time (seconds): 3.27
14 System time (seconds): 0.08
15 Percent of CPU this job got: 229%
16 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:01.46
17 Average shared text size (kbytes): 0
18 Average unshared data size (kbytes): 0
19 Average stack size (kbytes): 0
20 Average total size (kbytes): 0
21 Maximum resident set size (kbytes): 157060
22 Average resident set size (kbytes): 0
23 Major (requiring I/O) page faults: 0
24 Minor (reclaiming a frame) page faults: 33362
25 Voluntary context switches: 37
26 Involuntary context switches: 921
27 Swaps: 0
28 File system inputs: 0
29 File system outputs: 0
30 Socket messages sent: 0
31 Socket messages received: 0
32 Signals delivered: 0
33 Page size (bytes): 4096
34 Exit status: 0

```

Listing 272: Ausgabe Testfall 6Prompt 1Ausführung 5

```

1 Traceback (most recent call last):
2   File "/home/mauke/Documents/Uni/Bachelorarbeit/venv/lib/python3.13/site-
3     packages/pandas/core/indexes/base.py", line 3805, in get_loc
4     return self._engine.get_loc(casted_key)
5     ~~~~~
6   File "index.pyx", line 167, in pandas._libs.index.IndexEngine.get_loc
7   File "index.pyx", line 196, in pandas._libs.index.IndexEngine.get_loc
8   File "pandas/_libs/hashtable_class_helper.pxi", line 7081, in pandas._libs.
9     hashtable.PyObjectHashTable.get_item
10  File "pandas/_libs/hashtable_class_helper.pxi", line 7089, in pandas._libs.
11    hashtable.PyObjectHashTable.get_item
12  KeyError: 'Straftaten -insgesamt-'
13
14 The above exception was the direct cause of the following exception:
15
16 Traceback (most recent call last):
17   File "/home/mauke/Documents/Uni/Bachelorarbeit/testcases/testcase6/prompt2/
18     exec1/script.py", line 56, in <module>
19     unterbezirke_df['Straftaten -insgesamt-'] = pd.to_numeric(unterbezirke_df
20     ['Straftaten -insgesamt-'], errors='coerce').fillna(0)
21     ~~~~~
22   File "/home/mauke/Documents/Uni/Bachelorarbeit/venv/lib/python3.13/site-
23     packages/pandas/core/frame.py", line 4102, in __getitem__
24     indexer = self.columns.get_loc(key)
25   File "/home/mauke/Documents/Uni/Bachelorarbeit/venv/lib/python3.13/site-
26     packages/pandas/core/indexes/base.py", line 3812, in get_loc
27     raise KeyError(key) from err
28  KeyError: 'Straftaten -insgesamt-'
29
30 Command exited with non-zero status 1
31
32 Command being timed: "python testcase6/prompt2/exec1/script.py"
33
34 User time (seconds): 5.30
35
36 System time (seconds): 0.28
37
38 Percent of CPU this job got: 151%
39
40 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:03.69
41
42 Average shared text size (kbytes): 0
43
44 Average unshared data size (kbytes): 0
45
46 Average stack size (kbytes): 0
47
48 Average total size (kbytes): 0
49
50 Maximum resident set size (kbytes): 156392
51
52 Average resident set size (kbytes): 0
53
54 Major (requiring I/O) page faults: 0
55
56 Minor (reclaiming a frame) page faults: 171834
57
58 Voluntary context switches: 42
59
60 Involuntary context switches: 471
61
62 Swaps: 0
63
64 File system inputs: 0
65
66 File system outputs: 0
67
68 Socket messages sent: 0

```

```

42 Socket messages received: 0
43 Signals delivered: 0
44 Page size (bytes): 4096
45 Exit status: 1

```

Listing 273: Ausgabe Testfall 6 Prompt 2 Ausführung 1

```

1 Sheet Fallzahlen_2014 erfolgreich gelesen.
2 Sheet Fallzahlen_2015 erfolgreich gelesen.
3 Sheet Fallzahlen_2016 erfolgreich gelesen.
4 Sheet Fallzahlen_2017 erfolgreich gelesen.
5 Sheet Fallzahlen_2018 erfolgreich gelesen.
6 Sheet Fallzahlen_2019 erfolgreich gelesen.
7 Sheet Fallzahlen_2020 erfolgreich gelesen.
8 Sheet Fallzahlen_2021 erfolgreich gelesen.
9 Sheet Fallzahlen_2022 erfolgreich gelesen.
10 Sheet Fallzahlen_2023 erfolgreich gelesen.
11 Alle Sheets wurden erfolgreich zusammengeführt.
12 Anzahl der Zeilen nach Filtern: 1440
13 Traceback (most recent call last):
14   File "/home/mauke/Documents/Uni/Bachelorarbeit/venv/lib/python3.13/site-
      packages/pandas/core/indexes/base.py", line 3805, in get_loc
15     return self._engine.get_loc(casted_key)
16         ~~~~~^~~~~~
17   File "index.pyx", line 167, in pandas._libs.index.IndexEngine.get_loc
18   File "index.pyx", line 196, in pandas._libs.index.IndexEngine.get_loc
19   File "pandas/_libs/hashtable_class_helper.pxi", line 7081, in pandas._libs.
      hashtable.PyObjectHashTable.get_item
20   File "pandas/_libs/hashtable_class_helper.pxi", line 7089, in pandas._libs.
      hashtable.PyObjectHashTable.get_item
21 KeyError: 'Straftaten -insgesamt-'
22
23 The above exception was the direct cause of the following exception:
24
25 Traceback (most recent call last):
26   File "/home/mauke/Documents/Uni/Bachelorarbeit/testcases/testcase6/prompt2/
      exec2/script.py", line 62, in <module>
27     filtered_df['Straftaten -insgesamt-'] = filtered_df['Straftaten -
      insgesamt-'].astype(str).str.replace('.', '').astype(int)
28
29     ~~~~~^~~~~~
29   File "/home/mauke/Documents/Uni/Bachelorarbeit/venv/lib/python3.13/site-
      packages/pandas/core/frame.py", line 4102, in __getitem__
30     indexer = self.columns.get_loc(key)
31   File "/home/mauke/Documents/Uni/Bachelorarbeit/venv/lib/python3.13/site-
      packages/pandas/core/indexes/base.py", line 3812, in get_loc
32     raise KeyError(key) from err
33 KeyError: 'Straftaten -insgesamt-'
34 Command exited with non-zero status 1
35   Command being timed: "python testcase6/prompt2/exec2/script.py"
36   User time (seconds): 5.23
37   System time (seconds): 0.25
38   Percent of CPU this job got: 154%

```

```

39 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:03.55
40 Average shared text size (kbytes): 0
41 Average unshared data size (kbytes): 0
42 Average stack size (kbytes): 0
43 Average total size (kbytes): 0
44 Maximum resident set size (kbytes): 157808
45 Average resident set size (kbytes): 0
46 Major (requiring I/O) page faults: 0
47 Minor (reclaiming a frame) page faults: 172332
48 Voluntary context switches: 39
49 Involuntary context switches: 730
50 Swaps: 0
51 File system inputs: 0
52 File system outputs: 0
53 Socket messages sent: 0
54 Socket messages received: 0
55 Signals delivered: 0
56 Page size (bytes): 4096
57 Exit status: 1

```

Listing 274: Ausgabe Testfall 6Prompt 2Ausführung 2

```

1 Sheet 'Fallzahlen_2014' erfolgreich gelesen.
2 Sheet 'Fallzahlen_2015' erfolgreich gelesen.
3 Sheet 'Fallzahlen_2016' erfolgreich gelesen.
4 Sheet 'Fallzahlen_2017' erfolgreich gelesen.
5 Sheet 'Fallzahlen_2018' erfolgreich gelesen.
6 Sheet 'Fallzahlen_2019' erfolgreich gelesen.
7 Sheet 'Fallzahlen_2020' erfolgreich gelesen.
8 Sheet 'Fallzahlen_2021' erfolgreich gelesen.
9 Sheet 'Fallzahlen_2022' erfolgreich gelesen.
10 Sheet 'Fallzahlen_2023' erfolgreich gelesen.
11 Alle Sheets erfolgreich zusammengeführt.
12 Spaltennamen im kombinierten DataFrame:
13 ['LOR-Schlüssel (Bezirksregion)', 'Bezeichnung (Bezirksregion)', 'Straftaten
    \n-insgesamt-', 'Raub', 'Straßenraub,\nHandtaschen-raub', 'Körper-
    verletzen \n-insgesamt-', 'Gefährl. und schwere Körper-verletzung', '
    Freiheits-beraubung, Nötigung,\nBedrohung, Nachstellung', 'Diebstahl \n-
    insgesamt-', 'Diebstahl von Kraftwagen', 'Diebstahl \nan/aus Kfz', '
    Fahrrad-\ndiebstahl', 'Wohnraum-\neinbruch', 'Branddelikte \n-insgesamt-',
    'Brand-\nstiftung', 'Sach-beschädigung -insgesamt-', 'Sach-beschädigung
    durch Graffiti', 'Rauschgift-delikte', 'Kieztaten']
14 Bereinigte Spaltennamen:
15 ['LOR-Schlüssel (Bezirksregion)', 'Bezeichnung (Bezirksregion)', 'Straftaten
    -insgesamt-', 'Raub', 'Straßenraub, Handtaschen-raub', 'Körper-
    verletzen -insgesamt-', 'Gefährl. und schwere Körper-verletzung', '
    Freiheits-beraubung, Nötigung, Bedrohung, Nachstellung', 'Diebstahl -
    insgesamt-', 'Diebstahl von Kraftwagen', 'Diebstahl an/aus Kfz', 'Fahrrad
    - diebstahl', 'Wohnraum- einbruch', 'Branddelikte -insgesamt-', 'Brand-
    stiftung', 'Sach-beschädigung -insgesamt-', 'Sach-beschädigung durch
    Graffiti', 'Rauschgift-delikte', 'Kieztaten']
16 Anzahl der Unterbezirke nach Filter: 1570
17 /home/mauke/Documents/Uni/Bachelorarbeit/testcases/testcase6/prompt2/exec3/

```

```

script.py:53: UserWarning: This pattern is interpreted as a regular
expression, and has match groups. To actually get the groups, use str.
extract.
18 unterbezirke_df = unterbezirke_df[~unterbezirke_df['Bezeichnung (
    Bezirksregion)'].str.contains('|'.join(nicht_zugeordnet_keywords), case=
    False, na=False)]
19 Anzahl der Unterbezirke nach Entfernen nicht zugeordneter Zeilen: 1440
20 Traceback (most recent call last):
21   File "/home/mauke/Documents/Uni/Bachelorarbeit/testcases/testcase6/prompt2/
    exec3/script.py", line 59, in <module>
22     raise ValueError(f"Die Spalte '{straftaten_spalte}' wurde nicht gefunden.
    Bitte überprüfen Sie die Spaltennamen.")
23 ValueError: Die Spalte 'Straftaten -insgesamt-' wurde nicht gefunden. Bitte ü
    berprüfen Sie die Spaltennamen.
24 Command exited with non-zero status 1
25 Command being timed: "python testcase6/prompt2/exec3/script.py"
26 User time (seconds): 5.39
27 System time (seconds): 0.24
28 Percent of CPU this job got: 151%
29 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:03.72
30 Average shared text size (kbytes): 0
31 Average unshared data size (kbytes): 0
32 Average stack size (kbytes): 0
33 Average total size (kbytes): 0
34 Maximum resident set size (kbytes): 157240
35 Average resident set size (kbytes): 0
36 Major (requiring I/O) page faults: 0
37 Minor (reclaiming a frame) page faults: 172121
38 Voluntary context switches: 34
39 Involuntary context switches: 1016
40 Swaps: 0
41 File system inputs: 0
42 File system outputs: 0
43 Socket messages sent: 0
44 Socket messages received: 0
45 Signals delivered: 0
46 Page size (bytes): 4096
47 Exit status: 1

```

Listing 275: Ausgabe Testfall 6Prompt 2Ausführung 3

```

1 /home/mauke/Documents/Uni/Bachelorarbeit/testcases/testcase6/prompt2/exec4/
    script.py:61: SettingWithCopyWarning:
2 A value is trying to be set on a copy of a slice from a DataFrame.
3 Try using .loc[row_indexer,col_indexer] = value instead
4
5 See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
    stable/user_guide/indexing.html#returning-a-view-versus-a-copy
6 unterbezirke_df[straftaten_spalte] = pd.to_numeric(unterbezirke_df[
    straftaten_spalte], errors='coerce').fillna(0)
7
8      Unterbezirk  Straftaten_Gesamt
9 11      Berlin (PKS gesamt)      5269896
10 2      Alexanderplatz      202791

```

```

10 64      Lietzenburger Straße      122773
11 77      Moabit Ost      112148
12 128     Tempelhofer Vorstadt      106066
13 122     Südliche Friedrichstadt      95870
14 32      Frankfurter Allee Süd FK      95827
15 83      Neuköllner Mitte/Zentrum      91682
16 93      Otto-Suhr-Allee/Kantstraße      90443
17 131     Wedding Zentrum      90083
18 Command being timed: "python testcase6/prompt2/exec4/script.py"
19 User time (seconds): 5.26
20 System time (seconds): 0.23
21 Percent of CPU this job got: 154%
22 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:03.56
23 Average shared text size (kbytes): 0
24 Average unshared data size (kbytes): 0
25 Average stack size (kbytes): 0
26 Average total size (kbytes): 0
27 Maximum resident set size (kbytes): 159396
28 Average resident set size (kbytes): 0
29 Major (requiring I/O) page faults: 0
30 Minor (reclaiming a frame) page faults: 171976
31 Voluntary context switches: 36
32 Involuntary context switches: 566
33 Swaps: 0
34 File system inputs: 0
35 File system outputs: 0
36 Socket messages sent: 0
37 Socket messages received: 0
38 Signals delivered: 0
39 Page size (bytes): 4096
40 Exit status: 0

```

Listing 276: Ausgabe Testfall 6Prompt 2Ausführung 4

```

1 Traceback (most recent call last):
2   File "/home/mauke/Documents/Uni/Bachelorarbeit/venv/lib/python3.13/site-
   packages/pandas/core/indexes/base.py", line 3805, in get_loc
3     return self._engine.get_loc(casted_key)
4     ~~~~~^~~~~~
5   File "index.pyx", line 167, in pandas._libs.index.IndexEngine.get_loc
6   File "index.pyx", line 196, in pandas._libs.index.IndexEngine.get_loc
7   File "pandas/_libs/hashtable_class_helper.pxi", line 7081, in pandas._libs.
   hashtable.PyObjectHashTable.get_item
8   File "pandas/_libs/hashtable_class_helper.pxi", line 7089, in pandas._libs.
   hashtable.PyObjectHashTable.get_item
9 KeyError: 'Straftaten -insgesamt-'
10
11 The above exception was the direct cause of the following exception:
12
13 Traceback (most recent call last):
14   File "/home/mauke/Documents/Uni/Bachelorarbeit/testcases/testcase6/prompt2/
   exec5/script.py", line 40, in <module>
15     filtered_df['Straftaten -insgesamt-'] = pd.to_numeric(filtered_df['

```

```

Straftaten -insgesamt-'], errors='coerce')
16
~~~~~
17 File "/home/mauke/Documents/Uni/Bachelorarbeit/venv/lib/python3.13/site-
packages/pandas/core/frame.py", line 4102, in __getitem__
18 indexer = self.columns.get_loc(key)
19 File "/home/mauke/Documents/Uni/Bachelorarbeit/venv/lib/python3.13/site-
packages/pandas/core/indexes/base.py", line 3812, in get_loc
20 raise KeyError(key) from err
21 KeyError: 'Straftaten -insgesamt-'
22 Command exited with non-zero status 1
23 Command being timed: "python testcase6/prompt2/exec5/script.py"
24 User time (seconds): 5.18
25 System time (seconds): 0.23
26 Percent of CPU this job got: 152%
27 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:03.54
28 Average shared text size (kbytes): 0
29 Average unshared data size (kbytes): 0
30 Average stack size (kbytes): 0
31 Average total size (kbytes): 0
32 Maximum resident set size (kbytes): 158980
33 Average resident set size (kbytes): 0
34 Major (requiring I/O) page faults: 0
35 Minor (reclaiming a frame) page faults: 171255
36 Voluntary context switches: 34
37 Involuntary context switches: 214
38 Swaps: 0
39 File system inputs: 0
40 File system outputs: 0
41 Socket messages sent: 0
42 Socket messages received: 0
43 Signals delivered: 0
44 Page size (bytes): 4096
45 Exit status: 1

```

Listing 277: Ausgabe Testfall 6Prompt 2Ausführung 5

```

1 Bezeichnung (Bezirksregion) Straftaten -insgesamt-
2 0 Alexanderplatz 202791
3 1 Lietzenburger Straße 122773
4 2 Moabit Ost 112148
5 3 Tempelhofer Vorstadt 106066
6 4 Südliche Friedrichstadt 95870
7 5 Frankfurter Allee Süd FK 95827
8 6 Neuköllner Mitte/Zentrum 91682
9 7 Otto-Suhr-Allee/Kantstraße 90443
10 8 Wedding Zentrum 90083
11 9 Regierungsviertel 85584
12 Command being timed: "python testcase6/prompt3/exec1/script.py"
13 User time (seconds): 5.19
14 System time (seconds): 0.29
15 Percent of CPU this job got: 153%
16 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:03.56

```



```

17 Average shared text size (kbytes): 0
18 Average unshared data size (kbytes): 0
19 Average stack size (kbytes): 0
20 Average total size (kbytes): 0
21 Maximum resident set size (kbytes): 159916
22 Average resident set size (kbytes): 0
23 Major (requiring I/O) page faults: 0
24 Minor (reclaiming a frame) page faults: 171814
25 Voluntary context switches: 34
26 Involuntary context switches: 775
27 Swaps: 0
28 File system inputs: 0
29 File system outputs: 0
30 Socket messages sent: 0
31 Socket messages received: 0
32 Signals delivered: 0
33 Page size (bytes): 4096
34 Exit status: 0

```

Listing 278: Ausgabe Testfall 6 Prompt 3 Ausführung 1

```

1 Sheet 'Fallzahlen_2014' erfolgreich gelesen.
2 Sheet 'Fallzahlen_2015' erfolgreich gelesen.
3 Sheet 'Fallzahlen_2016' erfolgreich gelesen.
4 Sheet 'Fallzahlen_2017' erfolgreich gelesen.
5 Sheet 'Fallzahlen_2018' erfolgreich gelesen.
6 Sheet 'Fallzahlen_2019' erfolgreich gelesen.
7 Sheet 'Fallzahlen_2020' erfolgreich gelesen.
8 Sheet 'Fallzahlen_2021' erfolgreich gelesen.
9 Sheet 'Fallzahlen_2022' erfolgreich gelesen.
10 Sheet 'Fallzahlen_2023' erfolgreich gelesen.
11 Alle Sheets wurden erfolgreich zu einem einzigen DataFrame zusammengeführt.
12 Anzahl der verbleibenden Unterbezirke: 153
13 /home/mauke/Documents/Uni/Bachelorarbeit/testcases/testcase6/prompt3/exec2/
    script.py:57: SettingWithCopyWarning:
14 A value is trying to be set on a copy of a slice from a DataFrame.
15 Try using .loc[row_indexer,col_indexer] = value instead
16
17 See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
    stable/user_guide/indexing.html#returning-a-view-versus-a-copy
18 unterbezirke_df['Straftaten \n-insgesamt-'] = pd.to_numeric(unterbezirke_df
    ['Straftaten \n-insgesamt-'], errors='coerce')
19
20 Top 10 Unterbezirke mit den höchsten Gesamtstraftaten:
21 Bezeichnung (Bezirksregion)  Straftaten \n-insgesamt-
22 Alexanderplatz                202791
23 Lietzenburger Straße          122773
24 Moabit Ost                    112148
25 Tempelhofer Vorstadt          106066
26 Südliche Friedrichstadt       95870
27 Frankfurter Allee Süd FK      95827
28 Neuköllner Mitte/Zentrum      91682
29 Otto-Suhr-Allee/Kantstraße     90443

```

```

30      Wedding Zentrum                      90083
31      Regierungsviertel                    85584
32      Command being timed: "python testcase6/prompt3/exec2/script.py"
33      User time (seconds): 5.43
34      System time (seconds): 0.22
35      Percent of CPU this job got: 150%
36      Elapsed (wall clock) time (h:mm:ss or m:ss): 0:03.76
37      Average shared text size (kbytes): 0
38      Average unshared data size (kbytes): 0
39      Average stack size (kbytes): 0
40      Average total size (kbytes): 0
41      Maximum resident set size (kbytes): 159220
42      Average resident set size (kbytes): 0
43      Major (requiring I/O) page faults: 0
44      Minor (reclaiming a frame) page faults: 172058
45      Voluntary context switches: 34
46      Involuntary context switches: 578
47      Swaps: 0
48      File system inputs: 0
49      File system outputs: 0
50      Socket messages sent: 0
51      Socket messages received: 0
52      Signals delivered: 0
53      Page size (bytes): 4096
54      Exit status: 0

```

Listing 279: Ausgabe Testfall 6Prompt 3Ausführung 2

```

1      Unterbezirk  Gesamte Straftaten
2      2          Alexanderplatz          202791
3      74          Lietzenburger Straße    122773
4      87          Moabit Ost              112148
5      138         Tempelhofer Vorstadt    106066
6      132         Südliche Friedrichstadt 95870
7      42          Frankfurter Allee Süd FK 95827
8      93          Neuköllner Mitte/Zentrum 91682
9      103         Otto-Suhr-Allee/Kantstraße 90443
10     141          Wedding Zentrum         90083
11     114          Regierungsviertel       85584
12     Command being timed: "python testcase6/prompt3/exec3/script.py"
13     User time (seconds): 5.35
14     System time (seconds): 0.24
15     Percent of CPU this job got: 151%
16     Elapsed (wall clock) time (h:mm:ss or m:ss): 0:03.70
17     Average shared text size (kbytes): 0
18     Average unshared data size (kbytes): 0
19     Average stack size (kbytes): 0
20     Average total size (kbytes): 0
21     Maximum resident set size (kbytes): 159524
22     Average resident set size (kbytes): 0
23     Major (requiring I/O) page faults: 0
24     Minor (reclaiming a frame) page faults: 172236
25     Voluntary context switches: 34

```

```

26 Involuntary context switches: 282
27 Swaps: 0
28 File system inputs: 0
29 File system outputs: 0
30 Socket messages sent: 0
31 Socket messages received: 0
32 Signals delivered: 0
33 Page size (bytes): 4096
34 Exit status: 0

```

Listing 280: Ausgabe Testfall 6Prompt 3Ausführung 3

```

1           Unterbezirk  Gesamtstraftaten
2 2           Alexanderplatz           202791
3 74          Lietzenburger Straße      122773
4 87           Moabit Ost               112148
5 138         Tempelhofer Vorstadt       106066
6 132         Südliche Friedrichstadt    95870
7 42          Frankfurter Allee Süd FK   95827
8 93          Neuköllner Mitte/Zentrum   91682
9 103         Otto-Suhr-Allee/Kantstraße  90443
10 141         Wedding Zentrum           90083
11 114         Regierungsviertel         85584
12 Command being timed: "python testcase6/prompt3/exec4/script.py"
13 User time (seconds): 5.26
14 System time (seconds): 0.25
15 Percent of CPU this job got: 152%
16 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:03.62
17 Average shared text size (kbytes): 0
18 Average unshared data size (kbytes): 0
19 Average stack size (kbytes): 0
20 Average total size (kbytes): 0
21 Maximum resident set size (kbytes): 158204
22 Average resident set size (kbytes): 0
23 Major (requiring I/O) page faults: 0
24 Minor (reclaiming a frame) page faults: 172076
25 Voluntary context switches: 33
26 Involuntary context switches: 441
27 Swaps: 0
28 File system inputs: 0
29 File system outputs: 0
30 Socket messages sent: 0
31 Socket messages received: 0
32 Signals delivered: 0
33 Page size (bytes): 4096
34 Exit status: 0

```

Listing 281: Ausgabe Testfall 6Prompt 3Ausführung 4

```

1 Sheet 'Fallzahlen_2014' erfolgreich eingelesen.
2 Sheet 'Fallzahlen_2015' erfolgreich eingelesen.
3 Sheet 'Fallzahlen_2016' erfolgreich eingelesen.
4 Sheet 'Fallzahlen_2017' erfolgreich eingelesen.

```

```

5 Sheet 'Fallzahlen_2018' erfolgreich eingelesen.
6 Sheet 'Fallzahlen_2019' erfolgreich eingelesen.
7 Sheet 'Fallzahlen_2020' erfolgreich eingelesen.
8 Sheet 'Fallzahlen_2021' erfolgreich eingelesen.
9 Sheet 'Fallzahlen_2022' erfolgreich eingelesen.
10 Sheet 'Fallzahlen_2023' erfolgreich eingelesen.
11 Alle Sheets erfolgreich zu einem DataFrame kombiniert.
12 Anzahl der verbleibenden Unterbezirke: 153
13 /home/mauke/Documents/Uni/Bachelorarbeit/testcases/testcase6/prompt3/exec5/
    script.py:52: SettingWithCopyWarning:
14 A value is trying to be set on a copy of a slice from a DataFrame.
15 Try using .loc[row_indexer,col_indexer] = value instead
16
17 See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
    stable/user_guide/indexing.html#returning-a-view-versus-a-copy
18 unterbezirke_df['Straftaten \n-insgesamt-'] = pd.to_numeric(unterbezirke_df
    ['Straftaten \n-insgesamt-'], errors='coerce')
19
20 Top 10 Unterbezirke mit den höchsten Gesamtstraftaten:
21 Bezeichnung (Bezirksregion) Straftaten -insgesamt-
22 0 Alexanderplatz 202791
23 1 Lietzenburger Straße 122773
24 2 Moabit Ost 112148
25 3 Tempelhofer Vorstadt 106066
26 4 Südliche Friedrichstadt 95870
27 5 Frankfurter Allee Süd FK 95827
28 6 Neuköllner Mitte/Zentrum 91682
29 7 Otto-Suhr-Allee/Kantstraße 90443
30 8 Wedding Zentrum 90083
31 9 Regierungsviertel 85584
32 Command being timed: "python testcase6/prompt3/exec5/script.py"
33 User time (seconds): 5.23
34 System time (seconds): 0.26
35 Percent of CPU this job got: 154%
36 Elapsed (wall clock) time (h:mm:ss or m:ss): 0:03.56
37 Average shared text size (kbytes): 0
38 Average unshared data size (kbytes): 0
39 Average stack size (kbytes): 0
40 Average total size (kbytes): 0
41 Maximum resident set size (kbytes): 159796
42 Average resident set size (kbytes): 0
43 Major (requiring I/O) page faults: 0
44 Minor (reclaiming a frame) page faults: 171441
45 Voluntary context switches: 33
46 Involuntary context switches: 668
47 Swaps: 0
48 File system inputs: 0
49 File system outputs: 0
50 Socket messages sent: 0
51 Socket messages received: 0
52 Signals delivered: 0
53 Page size (bytes): 4096

```

54 `Exit status: 0`

Listing 282: Ausgabe Testfall 6Prompt 3Ausführung 5