

# Workflow-Dokumentation zur Geodatenverarbeitung von Sinus-Milieu-Daten in Kombination mit ALKIS- Adressen in Python

Begleitdokument zu den Python-Skripten  
<https://github.com/yannickelias/Sinus-Milieu>

## Impressum

Entstanden im Verbundvorhaben: WärmewendeNordwest - Digitalisierung zur Umsetzung von Wärmewende- und Mehrwertanwendungen für Gebäude, Campus, Quartiere und Kommunen im Nordwesten - Teilprojekte: Potentiale für eine stadtweite und quartiersbezogene Transformation der Wärmeversorgung sowie Bildungsformate für nachhaltige Entwicklung (FKZ: 03SF0624H).



### Zitiervorschlag:

Gerling, Y. & Schwarz, T. (2025): Workflow-Dokumentation zur Geodatenverarbeitung von Sinus-Milieu-Daten in Kombination mit ALKIS-Adressen in Python, DOI: [10.5281/zenodo.1488667](https://doi.org/10.5281/zenodo.1488667)

## Inhalt

<b>1</b>	<b>Hintergrund .....</b>	<b>4</b>
<b>2</b>	<b>Methodik.....</b>	<b>4</b>
<b>2.1</b>	<b>Adressstruktur-Anpassung.....</b>	<b>4</b>
<b>2.2</b>	<b>Sinus-Milieu-Aggregation.....</b>	<b>7</b>
<b>3</b>	<b>Schlussbemerkungen .....</b>	<b>10</b>
	<b>Quellen.....</b>	<b>10</b>

# 1 Hintergrund

Die vorliegende Kurzanleitung dient der Methodenbeschreibung zur Adressanpassung und Aggregation von gebäudescharfen Sinus-Milieu-Punktdaten z. B. auf Gitterzellen oder Baublöcke mit der Programmiersprache Python.

Grundlage der Bearbeitung war der Datensatz „MBM Sinus-Milieus® Haus für das Land Bremen auf der Gebäudeebene inkl. OSM Koordinaten und Hausadressen“ der Michael Bauer Micromarketing GmbH.

# 2 Methodik

Die Methodik mittels Python-Skripten unterteilt sich in zwei Hauptaufgaben: Das Bereinigen und Anpassen der vorliegenden Adressstrukturen sowie die Aggregation der Sinus-Milieus unter Gewichtung der angegebenen Haushaltsanzahl. Die nachfolgenden Beschreibungen verstehen sich als Ergänzung und Erläuterung der Python-Codes, die auf GitHub (<https://github.com/yannickelias/Sinus-Milieu.git>) bereitgestellt werden.

## 2.1 Adressstruktur-Anpassung

### Anpassen der ALKIS-Adressen:

Bevor die Verarbeitung beginnt, prüft der Code, ob die Spalte "LAGEBEZTXT", welche die Adressen enthält, in den eingelesenen Daten existiert. Falls diese fehlt, wird eine Fehlermeldung ausgegeben. Nun wird eine leere Liste (new\_rows) erstellt, um die neu generierten Adressdatensätze zu speichern. Danach beginnt eine Iteration über alle Zeilen der Geodaten. Dabei werden folgende Werte aus jeder Zeile extrahiert:

- `lagebez`: Der Inhalt der Adressspalte ("LAGEBEZTXT").
- `geometry`: Die geometrischen Koordinaten des Eintrags (z. B. Punkt- oder Flächendaten).
- `other_columns`: Alle weiteren Spalten außer "LAGEBEZTXT" und "geometry".

Falls eine Adresse vorhanden ist, wird sie zuerst anhand des Semikolons (;) in mehrere Adressen aufgeteilt. Anschließend wird jede dieser Adressen weiterverarbeitet. Falls eine Adresse eine Hausnummer enthält, wird sie durch das letzte Leerzeichen (`rsplit(" ", 1)`) getrennt, sodass Straßename und Hausnummern separat vorliegen.

Für jede einzelne Straße mit Hausnummer wird eine neue Adresse generiert, indem der Straßename mit der oder den jeweiligen Hausnummern kombiniert wird. Dann wird eine neue Zeile mit Adresse, Geometrie und weiteren Spalten erstellt und in die Liste new\_rows eingefügt. Falls eine Adresse keine Hausnummer enthält, wird sie direkt in die neue Datenstruktur übernommen. Nachdem alle Adressen verarbeitet wurden, wird ein neuer GeoDataFrame (new\_gdf) mit den aufgeteilten Daten erstellt, wobei die ursprüngliche Koordinatenreferenz (crs) beibehalten wird.

Abbildung 1 zeigt den Code-Ausschnitt dieser ersten Schritte.

```

1 #Trennung der unterschiedlichen Straßen in einem Feature, getrennt durch ein Simikolon:
2
3 # Eingabe- und Ausgabepfade
4 input_shapefile = r"C:\Users\ygerling\WWN\Geodaten\SINUS2024\Wohngeb_HB\Wohngeb_HB.shp"
5 output_shapefile = r"C:\Users\ygerling\WWN\Geodaten\SINUS2024\Wohngeb_HB\Ergebnis\Wohngeb_HB_Ergebnis"
6
7 # Shape-Datei einlesen
8 gdf = gpd.read_file(input_shapefile)
9
10 # Überprüfen, ob die Spalte "LAGEBEZTXT" existiert
11 if "LAGEBEZTXT" not in gdf.columns:
12     raise ValueError("Die Spalte 'LAGEBEZTXT' ist in der Eingabedatei nicht vorhanden.")
13
14 # Liste für die neuen Datenzeilen
15 new_rows = []
16
17 # Iteriere durch die Geodaten
18 for _, row in gdf.iterrows():
19     lagebez = row["LAGEBEZTXT"] # Adressspalte
20     geometry = row.geometry # Geometrie
21     other_columns = row.drop(["LAGEBEZTXT", "geometry"]) # Restliche Spalten
22
23     if lagebez:
24         # Adressen durch ";" trennen und bereinigen
25         addresses = [addr.strip() for addr in lagebez.split(";") if addr.strip()]
26
27         for address in addresses:
28             if " " in address:
29                 # Trenne Straße und Hausnummern
30                 street, house_numbers = address.rsplit(" ", 1)
31
32                 # Hausnummern durch "," trennen und bereinigen
33                 house_numbers_list = [hn.strip() for hn in house_numbers.split(",") if hn.strip()]
34
35                 for house_number in house_numbers_list:
36                     # Kombiniere Straße mit einzelner Hausnummer
37                     new_address = f"{street} {house_number}"
38                     new_row = other_columns.to_dict()
39                     new_row["LAGEBEZTXT"] = new_address
40                     new_row["geometry"] = geometry
41                     new_rows.append(new_row)

```

*Abb. 1: Python-Code-Abschnitt für die Anpassung der Adressstruktur*

Zunächst werden die Eingabe- und Ausgabepfade definiert. Eine leere Liste (new\_rows) wird erstellt, um die aufgeteilten Adressen zu speichern. Anschließend wird über jede Zeile der Datei iteriert. Dabei werden folgende Werte aus jeder Zeile extrahiert:

- lagebez: Die Adresse (Straße und Hausnummern) aus der Spalte "LAGEBEZTXT".
- geometry: Die geometrischen Daten (z. B. Koordinaten des Gebäudes).
- other\_columns: Alle anderen Spalten außer "LAGEBEZTXT" und "geometry".

Falls die Adressspalte Werte enthält, wird geprüft, an welcher Stelle die erste Ziffer vorkommt, um Straßenname und Hausnummer(n) zu trennen. Dazu wird eine Schleife durch den Zeichenstring (lagebez) durchgeführt:

- Sobald das erste Zeichen gefunden wird, das eine Ziffer (0-9) ist, wird dessen Position (split\_index) gespeichert.
- Falls keine Ziffer vorhanden ist, wird die Adresse als Straße ohne Hausnummern behandelt.

Nach der Identifikation der Trennstelle wird die Adresse in zwei Teile gespalten:

- street: Enthält den Straßennamen (alles vor der ersten Ziffer).
- house\_numbers: Enthält die Hausnummer(n) (alles ab der ersten Ziffer).
- Falls keine Hausnummern vorhanden sind, wird house\_numbers als leerer String gesetzt.

Falls Hausnummern vorhanden sind, werden sie durch Kommas (split(",")) getrennt. Dies ist notwendig, weil einige Gebäude mehrere Hausnummern haben, die gemeinsam in einer Zeile gespeichert sind (z. B. "15, 17, 19"). Für jede einzelne Hausnummer wird eine neue Adresse erstellt, indem der Straßename mit der jeweiligen Hausnummer kombiniert wird. Anschließend wird eine neue Zeile mit Adresse, Geometrie und weiteren Spalten erstellt und in die Liste new\_rows eingefügt. Abbildung 2 zeigt die beschriebenen Schritte im Python-Code.

```

15 # Iteriere durch die Zeilen der Shape-Datei
16 for _, row in gdf.iterrows():
17     lagebez = row["LAGEBEZTXT"]
18     geometry = row.geometry
19     other_columns = row.drop(["LAGEBEZTXT", "geometry"])
20
21     if lagebez:
22         # Adressentrennung mithilfe von isdigit()
23         split_index = None
24         for i, char in enumerate(lagebez):
25             if char.isdigit(): # Prüfe, ob das Zeichen eine Ziffer ist
26                 split_index = i
27                 break
28
29         # Trenne die Basisadresse (Straßenname) und die Hausnummern
30         if split_index is not None:
31             street = lagebez[:split_index].strip() # Alles vor der ersten Ziffer
32             house_numbers = lagebez[split_index:].strip() # Alles ab der ersten Ziffer
33         else:
34             # Wenn keine Ziffer gefunden wird
35             street = lagebez.strip()
36             house_numbers = ""
37
38         # Hausnummern durch ',' trennen
39         house_numbers_list = [hn.strip() for hn in house_numbers.split(",") if hn.strip()]
40
41         for house_number in house_numbers_list:
42             new_address = f"{street} {house_number}"
43             new_row = other_columns.to_dict()
44             new_row["LAGEBEZTXT"] = new_address
45             new_row["geometry"] = geometry
46             new_rows.append(new_row)
47

```

*Abb. 2: Python-Code-Abschnitt für die umstrukturierte Speicherung von Straße und Hausnummer bei vorliegenden Adressangaben*

#### Anpassen der Sinus-Adressen (vorliegend im GIS durchgeführt):

Der Angleich der Sinus-Adressdaten erfolgte abweichend zur bisherigen Vorgehensweise über den „Feld berechnen“ Operator in ArcGIS Pro. Grundsätzlich ist die Umsetzung hiervon aber auch analog mit dem Geodataframe in Python direkt möglich. Zunächst wurde ein neues Feld mit dem Namen AdresseSIN angelegt. Anschließend wurde für dieses Feld die „Feld berechnen“ Funktion verwendet, indem die beiden separaten Felder mit den Straßennamen und den Hausnummern addiert wurden:

AdresseSIN = !MB\_HA\_AD\_N! + !MB\_HA\_AD\_1!

Nun lagen die Adressen beispielhaft wie folgt vor: Aachener Str.10

Damit eine Verbindung (join) mit den ALKIS-Adressen erfolgen kann, wurde der Straßenname einheitlich von „Str.“ zu „Straße“ mittels folgender Codezeile geändert:

AdresseSIN = !AdresseSIN!.replace("str.", "straße ") und !AdresseSIN!.replace("Str.", "Straße ")

Abschließend liegen für beide Datensätze beispielhaft nun die Adressen folgendermaßen vor: Aachener Straße 10

Abbildung 3 zeigt die „Feld berechnen“ Funktion in ArcGIS Pro.

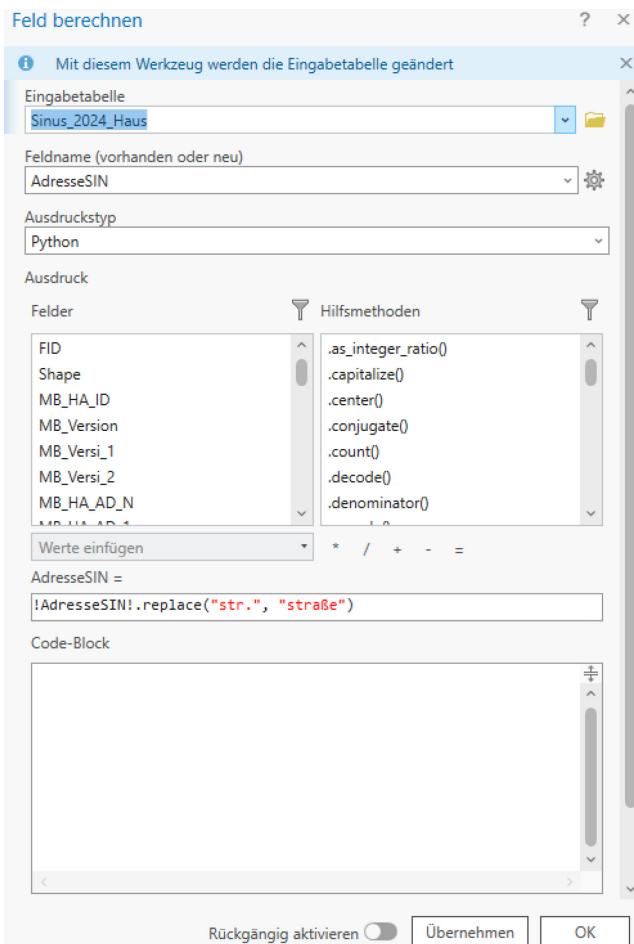


Abb. 3: Operation „Feld berechnen“ in ArcGIS Pro

#### Verbinden der beiden Datensätze anhand der Adressfelder:

Abschließend erfolgt das Verbinden der beiden Eingabe-Layer mittels einer Geooperation, die einen *Join* anhand der beiden Adressfelder durchführt. Für die weiterführende Bearbeitung wurden die Sinusdaten an die Gebäudebedaten des ALKIS angefügt.

## 2.2 Sinus-Milieu-Aggregation

Dieser Code dient der **räumlichen Aggregation** von Haushalts- und Milieudaten, indem er die Anzahl der Haushalte pro SINUS-Milieu innerhalb bestimmter Zielpolygone, wie Baublöcke oder Lambert-Raster, berechnet. Die Daten werden dabei aus einer Shape-Datei mit Haushaltsinformationen entnommen, anhand der geografischen Lage den jeweiligen Zielpolygonen zugeordnet und anschließend aggregiert. Das Ergebnis wird als neue Shape-Datei gespeichert, um eine strukturierte Darstellung der Haushaltsverteilung pro Milieu zu ermöglichen.

Zunächst wird sichergestellt, dass die Spalte "LAGEBEZTXT", die die Adressdaten enthält, tatsächlich in den Daten vorhanden ist. Anschließend werden die geometrischen Mittelpunkte (centroids) der Wohngebäude berechnet.

Diese Mittelpunkte dienen als repräsentative Punkte für die Zuordnung zu den Zielpolygonen. Der Code erstellt daher eine zusätzliche Geometriespalte, in der diese Mittelpunkte gespeichert werden.

Nun erfolgt die räumliche Verknüpfung (Spatial Join) der Gebäudemittelpunkte mit den Zielpolygonen. Dazu wird überprüft, in welchem Zielpolygon sich jeder Gebäude-Mittelpunkt befindet. Diese Zuordnung erfolgt mit der Geopandas-Funktion sjoin(), wodurch jeder Gebäude-Mittelpunkt die ID des zugehörigen Zielpolygons erhält. Dabei werden nur Gebäude berücksichtigt, die tatsächlich innerhalb eines Zielpolygons liegen.

Nach der erfolgreichen Zuordnung folgt die Aggregation der Haushalte nach SINUS-Milieu innerhalb jedes Zielpolygons. Hierzu werden die Daten nach Polygon-ID und Milieu-Typ gruppiert und innerhalb jeder Gruppe werden die Haushaltszahlen summiert. Diese aggregierten Werte werden in einer neuen Tabelle gespeichert, die später mit den Zielpolygonen zusammengeführt wird.

Im nächsten Schritt werden die aggregierten Daten mit den Zielpolygonen verknüpft, um sicherzustellen, dass jedes Polygon die entsprechenden Haushaltszahlen erhält. Falls ein Zielpolygon keine zugeordneten Haushaltsdaten hat, bleibt es trotzdem in der Ergebnistabelle erhalten, sodass eine vollständige Geodatenstruktur gewährleistet wird.

Abbildung 4 zeigt einen entsprechenden Python-Code-Abschnitt der beschriebenen Schritte.

```

8 # Feldnamen
9 household_field = "MB_HA_BA_A" # Anzahl der Haushalte
10 milieu_field = "MB_HA_SI_K" # Dominantes Milieu
11
12 # Shape-Dateien einlesen
13 print("Shape-Dateien werden eingelesen...")
14 input_polygons_with_data = gpd.read_file(input_polygons_with_data_path)
15 target_polygons = gpd.read_file(target_polygons_path)
16
17 # Berechne die Mittelpunkte der Eingabepolygone
18 print("Mittelpunkte der Eingabepolygone werden berechnet...")
19 input_polygons_with_data["centroid"] = input_polygons_with_data.geometry.centroid
20 input_centroids = input_polygons_with_data.set_geometry("centroid")
21
22 # Räumliche Verknüpfung: Mittelpunkte mit Zielpolygonen verknüpfen
23 print("Räumliche Verknüpfung wird durchgeführt...")
24 joined = gpd.sjoin(input_centroids, target_polygons, how="inner", predicate="within")
25
26 # Aggregation: Haushalte pro dominantem Milieu und Polygon
27 print("Aggregation wird durchgeführt...")
28 aggregation = (
29     joined.groupby(["index_right", milieu_field])[household_field]
30     .sum()
31     .reset_index()
32     .rename(columns={"index_right": "Polygon_ID", household_field: "Total_Households"})
33 )
34
35 # Aggregierte Daten mit den Zielpolygonen verknüpfen
36 print("Aggregierte Daten werden mit Zielpolygonen verknüpft...")
37 target_polygons["Polygon_ID"] = target_polygons.index # ID-Feld hinzufügen
38 aggregated_polygons = target_polygons.merge(aggregation, on="Polygon_ID", how="left")
39

```

*Abb. 4: Python-Code-Abschnitt für die Aggregation auf Zielpolygone*

Zuletzt wird die bereits aggregierte Shape-Datei (file\_path) eingelesen, welche die Haushaltszahlen (Total\_Hous) und Milieus (MB\_HA\_SI\_K) für die jeweiligen Zielpolygone enthält. Diese Daten wurden in einem vorherigen Schritt bereits mit den zugehörigen geografischen Informationen verbunden. Anschließend wird ein neuer DataFrame (df) erstellt, der nur die relevanten Spalten (BB\_Nr, MB\_HA\_SI\_K, Total\_Hous und geometry) enthält, um die Datenverarbeitung zu optimieren.

Um sicherzustellen, dass die Haushaltszahlen korrekt verarbeitet werden können, wird die Spalte Total\_Hous in einen numerischen Datentyp umgewandelt. Falls nicht numerische Werte auftreten, werden sie ignoriert und durch 0 ersetzt. Dies verhindert Fehler in den weiteren Berechnungen und stellt sicher, dass die Haushaltszahlen für jedes Milieu korrekt summiert werden.

## Ermitteln des dominanten Milieus pro Baublock

Zur Ermittlung des dominanten Milieus je Baublock wird für jede Baublock-ID (BB\_Nr) das Milieu mit der höchsten Haushaltszahl (Total\_Hous) bestimmt. Der entsprechende Eintrag wird in einem separaten DataFrame (df\_dominant) gespeichert, der nur die Baublock-ID und das dominante Milieu (Dominant\_Milieu) enthält.

Nach dieser Selektion erfolgt eine weitere Aggregation der Daten. Dabei werden für jeden Baublock:

- Alle vorhandenen SINUS-Milieus als kommagetrennte Zeichenfolge zusammengeführt.
- Die Haushaltszahlen der verschiedenen Milieus ebenfalls als kommagetrennte Werte gespeichert.
- Eine repräsentative Geometrie für den Baublock beibehalten.

Diese Aggregation erfolgt über die groupby()-Methode, die für jeden Baublock eine einzige Zeile erstellt, in der alle relevanten Informationen zusammengefasst werden. Das zuvor bestimmte dominante Milieu (df\_dominant) wird anschließend durch eine Verknüpfung (merge) mit den aggregierten Daten verbunden, sodass jeder Baublock eindeutig mit einem Hauptmilieu versehen wird.

Abbildung 5 zeigt den Python-Code dieser Schritte.

```

9 # Relevante Spalten auswählen
10 df = gdf[['BB_Nr', 'MB_HA_SI_K', 'Total_Hous', 'geometry']]
11
12 # Sicherstellen, dass 'Total_Hous' numerisch ist
13 df['Total_Hous'] = pd.to_numeric(df['Total_Hous'], errors='coerce').fillna(0)
14
15 # Dominantes Milieu für jeden Baublock bestimmen
16 df_dominant = df.loc[df.groupby('BB_Nr')['Total_Hous'].idxmax(), ['BB_Nr', 'MB_HA_SI_K']]
17 df_dominant = df_dominant.rename(columns={'MB_HA_SI_K': 'Dominant_Milieu'})
18
19 # Aggregation der Daten nach Baublock-ID
20 df_agg = df.groupby('BB_Nr').agg({
21     'MB_HA_SI_K': lambda x: ', '.join(x.astype(str)), # Alle Milieus in einer Zeile zusammenfassen
22     'Total_Hous': lambda x: ', '.join(x.astype(str)), # Haushaltszahlen entsprechend auflisten
23     'geometry': 'first' # Eine Geometrie für den Baublock übernehmen
24 }).reset_index()
25
26 # Das dominante Milieu hinzufügen
27 df_agg = df_agg.merge(df_dominant, on='BB_Nr', how='left')

```

Abb. 5: Python-Code zur Ermittlung der dominanten Milieus je Baublock

### 3 Schlussbemerkungen

Die vorliegende Code-Beschreibung dient lediglich als textuelle Ergänzung zur Dokumentation und Bereitstellung der entsprechenden Skripte auf <https://github.com/yannickelias/Sinus-Milieu.git>.

Die vorgestellten Arbeitsschritte sind an die Vorgehensweise aus Schwarz et al. (2023) angelehnt. Im Zuge der Aktualisierung der Sinus-Milieu-Struktur und Sinus-Datenbestände für Bremen trägt die vorgestellte Arbeit als Vorbereitung zu einer Überarbeitung der bisherigen Studie bei (Schwarz et al., 2025).

### Quellen

Schwarz, T., Kötterheinrich K. und Knies, J. (2023): Das Konzept der Sinus-Milieus für eine zielgruppen-spezifische Strategieentwicklung in der kommunalen Wärmeplanung am Beispiel Bremens, DOI: [10.26092/elib/2409](https://doi.org/10.26092/elib/2409)

Schwarz, T., Gerling, Y. und Knies, J. (2025, *in Arbeit*): Bürgerschaftliche Potenziale: Akteursidentifikation und -aktivierung in der Wärmewende - Aktualisierung der Sinus-Milieus® für Bremen [*unveröffentlicht*]

**Hochschule Bremen**  
City University of Applied Sciences



**KONTAKT**

**Hochschule Bremen**  
Neustadtswall 30  
28199 Bremen

[waermewende@hs-bremen.de](mailto:waermewende@hs-bremen.de)