OpenSchoolMaps: Abfrage und Visualisierung von OSM-Daten mit Python

OpenSchoolMaps.ch — Freie Lernmaterialien zu freien Geodaten und Karten

---

Ein Arbeitsblatt mit Übungen für Selbstlernende und Studierende.

Es gibt ein öffentliches **Repository** auf GitLab, das mit diesem Dokument verknüpft ist und Anschauungsmaterial sowie eine Übung enthält.

Überblick

Dieses Arbeitsblatt ist eine Einführung in OpenStreetMap (einschliesslich Overpass-API und der Overpass-Turbo-GUI). Weitere Informationen befinden sich auf OpenSchoolMaps.

Lernziele

Dieses Dokument bietet Lösungen und Tipps zu folgenden Fragen:

1. Wie kann man Daten aus OpenStreetMap (OSM) **extrahieren** und **geovisualisieren**?
2. Wie kann man OSM in einer interaktiven Karte darstellen?

Benötigte Zeit

Die Bearbeitung dieses Arbeitsblatts dauert ungefähr 40 Minuten, einschliesslich der Ersteinrichtung. Die tatsächliche Dauer hängt jedoch von den individuellen Vorkenntnissen ab.

Voraussetzungen

Das nachfolgend benötigte Jupyter Notebook kann auf drei Wegen initialisiert werden. Wir empfehlen, dieses über Binder zu starten. Der Link dazu befindet sich im öffentlichen GitLab-Repository. Alternativ befindet sich dort auch eine Anleitung, Jupyter Notebook über Docker zu starten. Wer es vorzieht, Jupyter Notebook mit Anaconda zu installieren, findet Links dazu im Kapitel Bibliografie und Ressourcen.

Einführung

Ein Beispiel – Outdoor-Tischtennistische!

Als Beispiel verwenden wir Outdoor-Tischtennistische aus der Geschichte “Zürich ist der coolste Ort in der Schweiz zum Tischtennisspielen (laut OpenStreetMap 😊)”

Abbildung 1. Tischtennis im Freien. (Quelle: www.concretesports.co.uk)

Was ist OpenStreetMap? – Eine kurze Einführung

OpenStreetMap (OSM) ist eine frei nutzbare, bearbeitbare Karte der gesamten Welt – ein gemeinschaftliches Projekt für Geodaten. Es ist die grösste Community im Bereich Vektor-Geodaten und nach Wikimedia die zweitgrösste Community insgesamt. Bei den Basiskarten und Points-of-Interest (POI) ist OSM mit Google Maps vergleichbar.

OSM umfasst Basiskarten, POIs, Routing und Geokodierung (z.B. Adressen).

Die Datenstruktur von OSM basiert auf Schlüssel-Wert-Paaren, sogenannten *Tags* (“NoSQL Schema”), und verwendet ein topologisches Vektor-Geometrie-Modell bestehend aus *Nodes*, *Ways* und *Relations* (siehe die Übersicht und weiterführenden Unterseiten dazu im OSM-Wiki):

- Node: osm_id, lat/lon-Koordinaten, Reihe von Tags.
- Way: osm_id, Liste von Node osm_ids (Fremdschlüssel), Reihe von Tags.

Diese OSM-Datenstruktur unterscheidet sich stark von traditionellen Geoinformationssystemen (GIS) mit Ebenen und von relationalen Datenbanken mit Tabellen.

- Es gibt Punkte, Linien und Polygone. Polygone sind entweder geschlossene Ways oder aber Relations mit einem entsprechenden Tag 'area'.
- Tags sind Attribute, die für jedes Objekt der realen Welt variieren können.

Details werden schrittweise den Objekten (Entitäten) hinzugefügt. Es gibt keine vordefinierte *konzeptionelle Modellierung* einer *Entität* (bzw. Entitätsmenge) wie in GIS.

Beispiele:

- Gebäude: Können — nebst der Geometrie — aus nur einem Tag `building=yes` bestehen oder aber ein Restaurant sein mit dutzenden Tags.
- Kirche: Kann ein Gebäude sein, ein historischer Ort, ein Gottesdienstort usw.
- Bad: Kann ein privater Pool, ein öffentliches Schwimmbad, ein Park mit Badeeinrichtungen, ein Wasserpark usw. sein.

Beispiel für eine Overpass-Abfrage, die versucht, alle Badeobjekte in OSM zu erfassen: https://osm.li/COe

```
[out:json];
area["name"="Schweiz/Suisse/Svizzera/Svizra"];
(
  nwr[sport=swimming][name](area);
  nwr[leisure=swimming_area][name];
  nwr[leisure=water_park];
  nwr[amenity=public_bath];
);
out center;
```

Extrahieren, Verarbeiten und Geovisualisieren von OSM-Daten auf einer Karte

Wir wollen nun bestimmte Daten aus OpenStreetMap extrahieren und auf einer Karte ausgeben: Visualisieren wir alle Tischtennistische in Zürich. Dazu suchen wir zuerst den Tag unserer Daten heraus.

1. Laden von OSM-Daten
2. Verarbeitung und Speicherung als GeoJSON-Datei
3. Geovisualisierung als interaktive Karte mit einer Basiskarte

Zunächst wird der *Tag* für Tischtennistische bestimmt: Es ist `sport=table_tennis`. Dies lässt sich mit der Overpass-Turbo-GUI überprüfen: https://osm.li/VxG

```
/*
Table Tennis Tables
*/
[out:json];
node["sport"="table_tennis"]({{bbox}});
out body; >; out skel qt;
```

Zudem benötigen wir die Bounding Box des Gebiets der Stadt Zürich.

Eine Bounding Box ist ein rechteckiger Bereich, der ein Gebiet umgibt. bboxfinder oder das BBOX Webtool sind Werkzeuge, um die Koordinaten von gewünschten Orten zu ermitteln.

Interaktive Karte in einem Jupyter Notebook

Jupyter Notebooks sind webbasierte interaktive Softwareumgebungen, mit denen Notebook-Dokumente erstellt werden können. Ein Jupyter Notebook trägt die Dateiendung `.ipynb` und enthält eine Liste von Eingabe- und Ausgabezellen, die Code, Text und Diagramme (z.B. Karten) enthalten können.

Veröffentlichung der Karte im Web

Die mit den Bibliotheken Folium oder Plotly erstellten Karten können auch im Web veröffentlicht werden.

Dazu generiert man mit der "Export"-Funktion der jeweiligen Bibliothek eine HTML5-Datei (einschliesslich JavaScript und CSS). Das Veröffentlichen geht ohne Backend, d.h. man muss nur die generierten Dateien irgendwo hosten.

Wir werden Ihnen nun Anwendungsbeispiele im Jupyter Notebook zeigen. Folgen Sie dazu dem Link zum GitLab-Repository und lesen Sie dort das README.md.

Fazit und Ausblick

Wir benutzten dieses Python-Skript, um Daten von Overpass zu laden und auf einer Karte auszugeben.

```
import plotly.express as px
import geopandas as gpd
from collections import namedtuple
from osm2geojson import overpass_call, json2geojson

# BBox mit den Komponenten in der Reihenfolge, die von Overpass erwartet wird
Bbox = namedtuple("Bbox", ["south", "west", "north", "east"])
Tag = namedtuple("Tag", ["key", "value"])

# Deklaration der EPSG-Codes für verschiedene Koordinatenreferenzsysteme (crs)
WGS84 = 4326
webMercator = 3857

def load_osm_from_overpass(bbox, tag, crs=f"epsg:{WGS84}") -> gpd.GeoDataFrame:
    geojson = load_osm_from_overpass_geojson(bbox, tag)
    return gpd.GeoDataFrame.from_features(geojson, crs=crs)

def load_osm_from_overpass_geojson(bbox, tag):
    query = f"""
    [out:json];
    nwr["{tag.key}"="{tag.value}"]({bbox.south},{bbox.west},{bbox.north},{bbox.east});
```

```
    out body;
    >;
    out skel qt;
    """
    response = overpass_call(query)
    return json2geojson(response)

def plot_gdf_on_map(gdf, title=""):
    fig = px.scatter_map(
        gdf,
        lat=gdf.geometry.y,
        lon=gdf.geometry.x,
        color_discrete_sequence=["fuchsia"],
        height=500,
        zoom=11,
    )
    fig.update_layout(title=title, map_style="open-street-map")
    fig.update_layout(margin={"r": 0, "l": 0, "b": 0})
    return fig

def main():
    # Beispiel: Tischtennistische in Zürich
    tag = Tag(key="sport", value="table_tennis")
    bbox_zurich = Bbox(west=8.471668, south=47.348834, east=8.600454, north=47.434379)
    table_tennis_gdf = load_osm_from_overpass(bbox_zurich, tag)
    # Konvertieren aller Geometrien zu Punkten
    # (siehe https://gis.stackexchange.com/questions/302430/polygon-to-point-in-
geopandas)
    # Um Berechnungsfehler zu minimieren, konvertieren wir zu EPSG:3857 (webMercator)
    # und danach wieder zurück zu EPSG:4326 (WGS84)
    gdf = table_tennis_gdf.to_crs(epsg=webMercator)
    gdf.geometry = gdf.geometry.centroid
    gdf = gdf.to_crs(epsg=WGS84)
    fig = plot_gdf_on_map(gdf, title="Zurich Table Tennis")
    # Falls kein Fenster erscheint, die folgende Zeile auskommentieren
    # und die Datei standalone_map.html im Browser öffnen:
    # fig.write_html("standalone_map.html")
    fig.show()

if __name__ == "__main__":
    main()
```

Listing 1: Das Python-Skript, welches alle Tischtennistische in Zürich auf einer Karte anzeigt.

Die Aufgabe war es, das Skript so zu ändern, dass alle Trinkbrunnen in Rapperswil-Jona angezeigt werden. Die Lösungen dazu befinden sich in diesem Zip, auch auffindbar in der Tutorialübersicht.

Das Ergebnis visualisiert mit Plotly

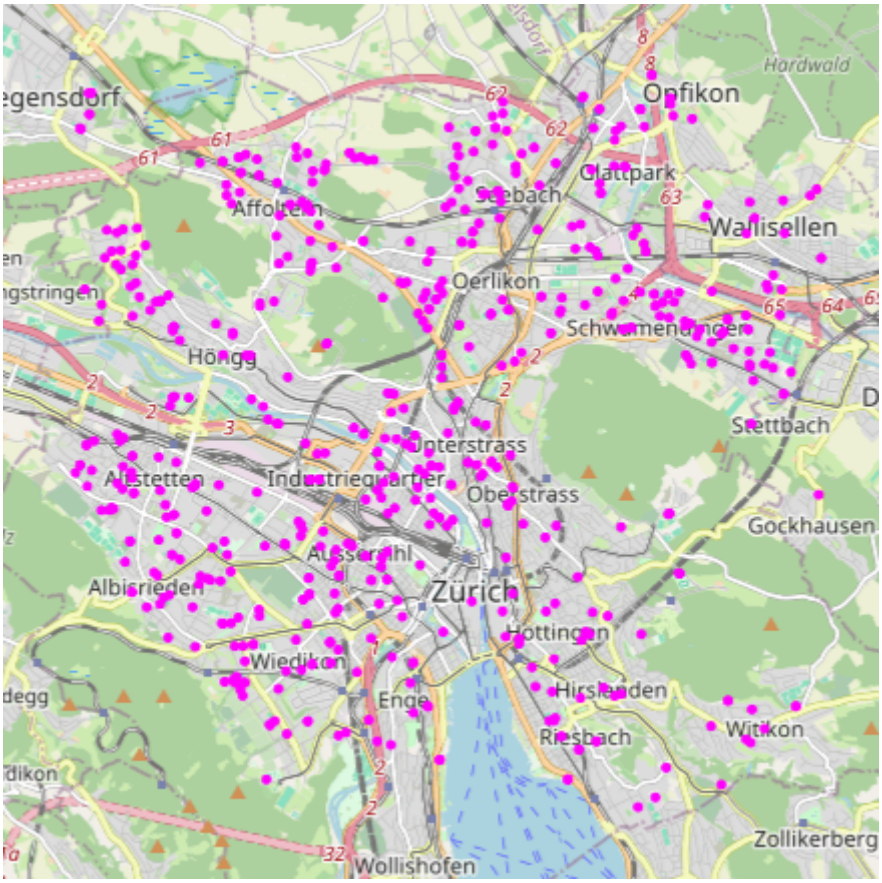

Abbildung 2. Karte der Tischtennistische in Zürich. Screenshot einer interaktiven Applikation, erstellt mit Python und Plotly. (Quelle: Eigene Arbeit)

Neben Plotly (und Folium) ist auch Matplotlib weit verbreitet. Für Python-Programmierer sind ausserdem drei weitere Möglichkeiten zu erwähnen, mit denen man Karten erstellen kann:

Streamlit + Plotly/MapLibre/Leafmap

In Streamlit kann man Plotly-Karten direkt anzeigen (`st.plotly_chart()`) und MapLibre-/Leafmap-Karten über eine Komponente bzw. `to_streamlit()` nahtlos integrieren.

Python + MapLibre

Wenn mehr Kontrolle über Styles benötigt wird, kann MapLibre GL JS über eine Streamlit-Komponente (HTML/JS) aus Python heraus eingebettet werden, um basierend auf Vektor-Tiles performante WebGL-Karten zu erstellen.

Python + Leafmap

Leafmap rendert Karten ohne eigenes JavaScript, bietet eine High-Level-Python-API und ist zudem GeoPandas-freundlich.

Bibliografie und Ressourcen

Installation der Jupyter-Software unter Windows:

1. Anaconda — software distribution tool

2. Jupyter-Installation (weitere Infos: https://mas-dse.github.io/startup/anaconda-windows-install/ , https://jupyterlab.readthedocs.io/en/stable/getting_started/installation.html)

Jupyter Notebooks / Anleitungen:

- Verschiedene Open-Source-Tools zur Geodatenvisualisierung in Jupyter Notebooks (Python): https://medium.com/@bartomolina/geospatial-data-visualization-in-jupyter-notebooks-ffa79e4ba7f8
- OSMnx ist eine Python-Bibliothek zum einfachen Herunterladen und Visualisieren von OpenStreetMap-Daten, deren Github-Repository Tutorials und Funktionsdemonstrationen beinhaltet.
- "A Guide: Turning OpenStreetMap Location Data into ML Features — How to pull shops, restaurants, public transport modes and other local amenities into your ML models." von Daniel L J Thomas, 17. September 2020. https://towardsdatascience.com/a-guide-turning-openstreetmap-location-data-into-ml-features-e687b66db210 (Hinweis: Beinhaltet OSMnx, Overpass, K-D-Tree zur Distanzberechnung)

Softwarebibliotheken und Frameworks für OSM und Python:

- Python Library "OSMnx" (Liniennetzwerke, POI): https://osmnx.readthedocs.io/en/stable/osmnx.html#module-osmnx.pois and https://autogis-site.readthedocs.io/en/latest/lessons/lesson-6/retrieve-data-from-openstreetmap.html
- Weitere Python Libraries:
  - osm2geojson: https://github.com/aspectumapp/osm2geojson
  - OverPy: https://github.com/DinoTools/python-overpy
- "Awesome OpenStreetMap" — Eine kuratierte Liste von'`allem' rund um OpenStreetMap (Noch nicht vollständig): https://github.com/osmlab/awesome-openstreetmap#python

Overpass API und GUI:

- Overpass API: https://medium.com/data-science/loading-data-from-openstreetmap-with-python-and-the-overpass-api-513882a27fd0
- Nutzung der Overpass API: https://gis.stackexchange.com/questions/203300/how-to-download-all-osm-data-within-a-boundingbox-with-overpass
- Beispiel für eine typische Overpass-Abfrage:

```
[out:json];
area["ISO3166-2"="CH-ZH"];
(
  nwr[sport="table_tennis"](area);
  nwr[leisure="table_tennis_table"];
);
out center;
```

Noch Fragen? Wenden Sie sich an OpenStreetMap Schweiz oder Stefan Keller!