Abfrage und Visualisierung von OSM-Daten mit Python

Ein Arbeitsblatt mit Übungen für Selbstlernende und Studierende.

Es gibt ein öffentliches **Repository** auf GitLab, das mit diesem Dokument verknüpft ist.

1. 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?
3. Wie kann man eigene Daten mit **Adressen** und **Ortsnamen** in Koordinaten umwandeln (Geokodierung)?
4. Wie kann man mit einem **Spatial Join** eigene Daten mit offenen geografischen Daten verbinden?

Benötigte Zeit

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

Voraussetzungen

Folgende Software sollte installiert sein (siehe Bibliografie und Ressourcen unten):

- Python 3
- Jupyter Notebook
- Einige Software-Bibliotheken (siehe `environment.yml` im Jupyter Notebook)

2. 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 😊)"

Figure 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*:

- 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:

```
[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

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

Lösung mit ~50 Zeilen Python-Code

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;
```

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

# BBox mit den Komponenten in der Reihenfolge, die von overpass.ql_query erwartet wird
Bbox = namedtuple("Bbox", ["south", "west", "north", "east"])
```

```
Tag = namedtuple("Tag", ["key", "values"])

def load_osm_from_overpass(bbox, tag, crs="epsg:4326") -> gpd.GeoDataFrame:
    geojson = load_osm_from_overpass_geojson(bbox, tag, crs)
    return gpd.GeoDataFrame.from_features(geojson, crs=crs)

def load_osm_from_overpass_geojson(bbox, tag, crs="epsg:4326"):
    values = None if "*" in tag.values else tag.values
    query = f"""
    [out:json];
    node["{tag.key}"="{values[0] if values else
'*'}"]({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, color="k", title=""):
    fig = px.scatter_mapbox(
        gdf,
        lat=gdf.geometry.y,
        lon=gdf.geometry.x,
        color_discrete_sequence=["fuchsia"],
        height=500,
        zoom=11,
    )
    fig.update_layout(title=title, mapbox_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", values=["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)
    gdf = table_tennis_gdf.to_crs(epsg=3857)
    gdf.geometry = gdf.geometry.centroid
    gdf = gdf.to_crs(epsg=4236)
    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: Der Jupyter Notebook-Skript in Python.

Aufgabe: Ändern Sie das Skript, um alle Trinkbrunnen in Rapperswil-Jona anzuzeigen. Finden Sie den passenden Tag und passen Sie die Bounding Box (Bbox) entsprechend an.

Sie finden die Lösung in der Datei `loesungen.py` entweder im `loesungen`-Verzeichniss oder aber auf OpenSchoolMaps.

Eine Bounding Box ist ein rechteckiger Bereich, der ein Gebiet umgibt. Sie können bboxfinder oder alternativ BBOX Webtool verwenden, um die Koordinaten von Rapperswil (SG) zu erhalten.

Das Ergebnis visualisiert mit Matplotlib

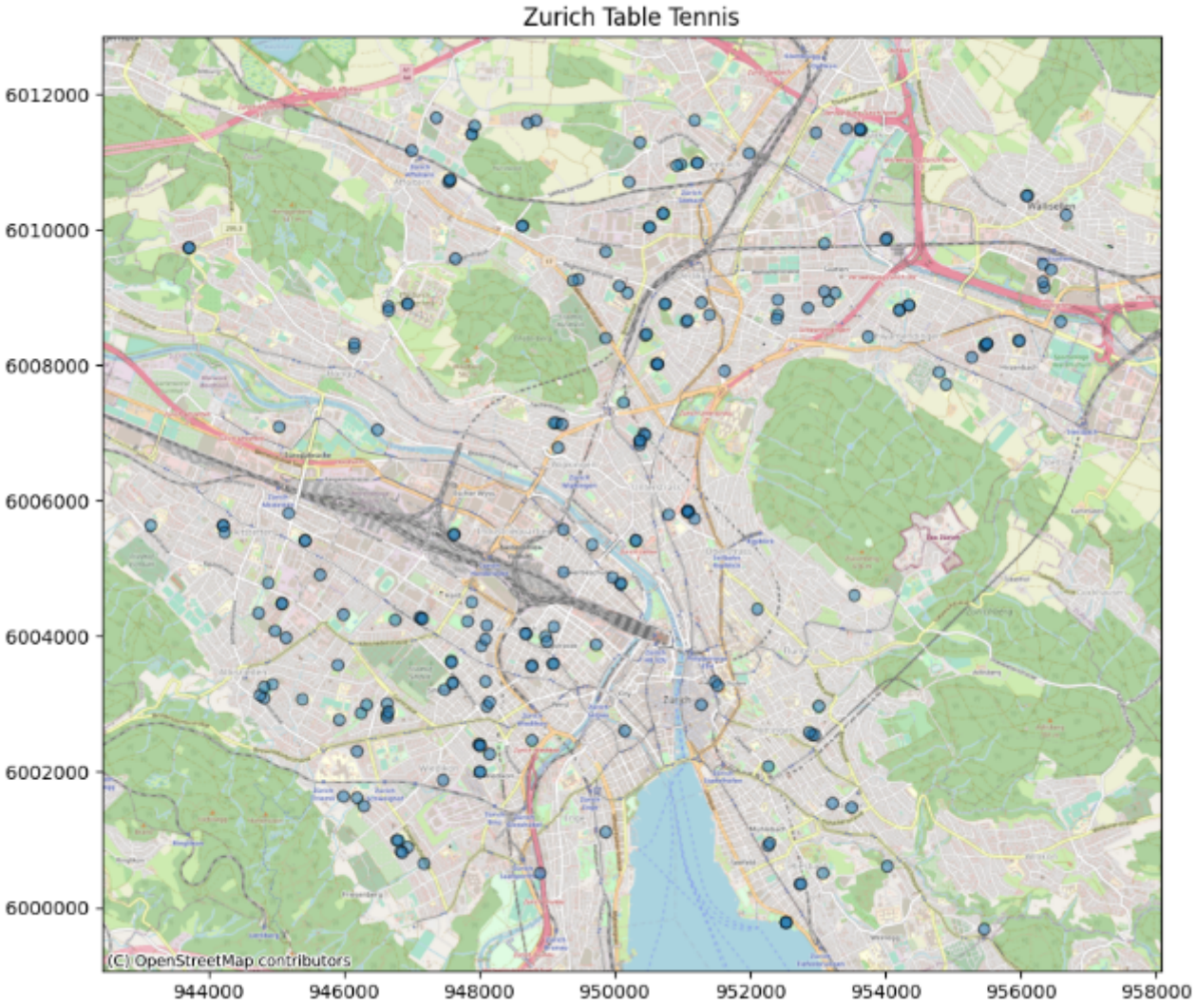

Figure 2. Karte der Tischtennistische in Zürich, erstellt mit Python und Matplotlib. (Quelle: Eigene Arbeit)

Interaktive Karte in einem Jupyter Notebook

Der Python-Code kann wie oben verwendet werden. Man muss ihn lediglich direkt aufrufen, anstatt `main()` zu verwenden.

Jupyter Notebooks mit der Bibliothek Folium

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

Es ist sehr einfach, ein Notebook-Dokument im kostenlosen Webdienst MyBinder auszuführen. Alternativ kann man Jupyter Notebooks auch selbst hosten, z.B. mit Anaconda.

Veröffentlichung der Karte im Web

Als Nächstes wird die im Jupyter Notebook erstellte Karte im Web veröffentlicht – ohne Backend, d.h. als generierte HTML5-Datei (einschliesslich JavaScript und CSS). Dies kann mit der "Export"-Funktion von Folium erfolgen. Siehe dazu die Lösungen.

Man kann Matplotlib auch in eine HTML5-Karte umwandeln mit diesem Code: Convert matplotlib plots from Python into interactive Leaflet web maps.

Die “magischen 3 Schritte” der Standortanalyse

Die "magischen 3 Schritte" der Standortanalyse bestehen aus der Kombination eigener Daten mit externen offenen Daten. Dieser sogenannte “Geographic Data Enrichment”-Prozess (hier ein Beitrag) ist möglich, da der Standort als universeller “universeller Fremdschlüssel” fungiert.

Ein wesentlicher Bestandteil dieses Prozesses ist die Anreicherung eigener Daten – die wahrscheinlich Ortsnamen oder Adressen enthalten – durch sogenannte *Geokodierung*, ein wichtiges Thema für sich.

Geokodierung bedeutet, einer postalischen Adresse oder einem geografischen Namen eine Koordinate zuzuweisen.

Gegeben sind Ortsnamen oder Adressdaten, dann folgt:

1. Die Geokodierung der eigenen Daten, die eine Adresse oder einen Ortsnamen enthalten sollten (d.h. Hinzufügen einer Koordinatenspalte).
2. Die räumliche Verknüpfung mit externen offenen Geodaten (z.B. Tischtennistische) mithilfe von GIS-Software (z.B. PostGIS).
3. Die anschliessende Analyse oder Geovisualisierung der Daten.

Dies wird anhand von PostgreSQL mit der Erweiterung PostGIS und der integrierten Sprache PL/pgSQL sowie Python demonstriert.

Eine einfache Alternative zur Geokodierung – neben der hier gezeigten Methode mit Python und PostgreSQL/PostGIS – wäre die Nutzung dieses Online-Dienstes von Localfocus.nl. Für kommerzielle Geokodierungsdienste siehe auch “WIGeoGIS”.

Schritt 1: Geokodierung von Adressen und Ortsnamen

Hier betrachten wir den Kern einer Geokodierungsfunktion in Python, die in eine PostgreSQL-Funktion in der PL/pgSQL-Sprache eingebettet ist.

Falls `CREATE LANGUAGE plpython3u;` fehlschlägt, installiere Python 3 für PostgreSQL wie folgt:

- Siehe https://www.python.org/downloads/. Alternativ kann QGIS 3 installiert werden, das Python bereits enthält.
- Die Systemumgebungsvariablen `PATH` und `PYTHONPATH` müssen auf die installierten Python-Bibliotheken verweisen. Dies kann durch Bearbeiten oder Hinzufügen der Variablen erfolgen (einmalig für alle Benutzerkonten). Unter Windows geht dies über **start** in the lower left corner and **Edit system environment variables**. Example (Windows):

```
PATH=$PATH$;C:\Program Files\QGIS 3.8\apps\Python37
PYTHONPATH=C:\Program Files\QGIS 3.8\apps\Python37
```

- Danach muss der PostgreSQL-Server neu gestartet werden. Siehe z.B. https://tableplus.com/blog/2018/10/how-to-start-stop-restart-postgresql-server.html. Beispiel für Windows:
  - Öffne die Eingabeaufforderung als Administrator
  - Öffne das Ausführen-Fenster mit `Win` + `R`
  - Tippe `services.msc` ein.
  - Suche den PostgreSQL-Dienst entsprechend der installierten Version.
  - Klicke auf “stop”, “start” oder “restart the service”.

Füge die fehlenden Parameter hinzu, um die Koordinaten für `Vulkanstrasse 106, Zürich` zu erhalten.

Siehe auch die Dokumentation der Nominatim-API.

```
CREATE LANGUAGE plpythonu;

CREATE OR REPLACE FUNCTION geocode(address text)
RETURNS text AS $$
"""Returns WKT geometry which is EMPTY if address was not found.

Please respect the Nominatim free usage policy:
https://operations.osmfoundation.org/policies/nominatim/
"""

import requests
from time import sleep

base_url = 'https://nominatim.openstreetmap.org/search?'
params = {
    'q': address,
    'format': 'json',
    'limit': 1,
    'User-Agent': 'geocode hack (pls. replace)'
}
response = requests.get(base_url, params=params)
loc = response.json()
if len(loc) == 0:
    wkt = 'SRID=4326;POINT EMPTY'
else:
    wkt = f'SRID=4326;POINT({loc[0]['lon']} {loc[0]['lat']})'
sleep(1)  # Throttle, to avoid overloading Nominatim
return wkt
$$ LANGUAGE plpythonu VOLATILE COST 1000;

-- Test:
SELECT ST_AsGeoJSON(geocode('Vulkanstrasse 106, Zürich'));
```

Discussion:

- Dies ist nur der Code. Er wurde noch nicht ausgeführt. Dies wird in Schritt zwei weiter unten als einzelne SQL-Abfrage erfolgen (Beachte dabei die Nominatim-Richtlinien für die freie Nutzung).
- WKT -"Well Known Text" zur Kodierung von Geometrien; Beispiel: `POINT(47.36829, 8.54836)`.
- `SRID=4326` - Spatial Reference ID, eine Kennung für das Koordinatenreferenzsystem WGS84, das für Breitengrad/Längengrad-Koordinaten verwendet wird.
- Ersetze bitte `geocode hack` im `User-Agent` durch einen spezifischen Namen für deinen Client. Andernfalls riskierst du, von der Nominatim-API gesperrt zu werden.

Schritt 2: Räumliche Verknüpfung

Nun, da die Geokodierungsfunktion implementiert ist, wird eine **räumliche Verknüpfung** durchgeführt. Das bedeutet, dass die Daten – zum Beispiel drei Adressen – mit externen Geodaten

wie beispielsweise Tischtennistische kombiniert werden, indem eine Geodatenbank wie PostGIS verwendet wird.

Wir nutzen hierfür eine OSM-Schweiz-Datenbank, die mit dem Tool osm2pgsql in PostgreSQL/PostGIS importiert wurde.

Ziel der Analyse: **Liste der fünf nächstgelegenen Tischtennistische mit zugehörigen 'customers'. Output:** `customer_id`, `distance`, `osm_id` + `osm_type` **(Node, Way, Relation), geomGeometrieetry.**:

- 1: Vulkanstrasse 106, Zürich
- 2: Bellevue, 8001 Zürich
- 3: Weihnachtsmann

```
WITH my_customers_tmp(id, address, geom) AS (
  SELECT id, address, ST_Transform(ST_MakeValid(GeomFromEWKT(geocode(address))),3857)
AS geom
  FROM (values
   (1,'Vulkanstrasse 106, Zürich'),
   (2,'Bellevue, 8001 Zürich'),
   (3,'Weihnachtsmann')
  ) address(id,address)
),
poi_tmp AS (
  SELECT
    osm_id,
    gtype,
    -- Die Adresse des POI müsste rückgeokodiert werden!
    tags,
    way AS geom FROM osm_poi
  WHERE tags->'sport' IN ('table_tennis')
)
SELECT
  my_customers_tmp.id as customer_id,
  ST_Distance(my_customers_tmp.geom, poi_tmp.geom)::integer AS distance,
  poi_tmp.osm_id,
  poi_tmp.gtype,
  ST_Transform(poi_tmp.geom,4326) AS geom
FROM my_customers_tmp, poi_tmp
ORDER BY my_customers_tmp.geom <-> poi_tmp.geom
LIMIT 5;
```

Listing 3: SQL-Abfrage des Geocode-Hacks mit Nominatim API in Python und PostgreSQL/PostGIS.

Diskussion:

- Dies ist eine moderne SQL-Abfrage, die eine Common Table Expression (`WITH` statement) verwendet. Sie speichert das Geokodierungsergebnis in einer temporären Tabelle `my_customers_tmp`.

- Die zweite temporäre Tabelle `poi_tmp` enthält die Tischtennistische und führt Abfragen auf der Basistabelle `osm_poi` aus. Diese ist eine Ansicht von `osm_point` und `osm_polygon` aus einer OSM-Datenbank mit Schweizer Daten (`osm_poi` ist somit vergleichbar mit der "nwr und center"-Abfrage in der Overpass-Abfrage zuvor).
- Die Anmerkung “POI address would need to reverse_geocode!” soll darauf hinweisen, dass man möglicherweise Adressen für die gefundenen POIs – hier: Tischtennistische – ermitteln möchte.

Schritt 3: Datenanalyse und Visualisierung

Das folgende Ergebnis zeigt die räumliche SQL-Abfrage:

	customer_id integer	distance integer	osm_id bigint	gtype text	geom geometry
1	2	438	553924669	po	0101000020E6100000BB8A1A4CC3182140AB7EC9EF23AF4740
2	1	693	5536222518	pt	0101000020E6100000C9D23DA18DFA2040A93551CFCCB24740
3	2	783	6390501170	pt	0101000020E610000085BB0E304E1721405DCC950F9CAF4740
4	1	861	617302896	pt	0101000020E6100000027855B142FB2040A0350852CEB14740
5	1	977	5075479505	pt	0101000020E6100000C071BE7DC2F620402B33ECC5ABB14740

Figure 3. Resultat der Geokodierung mit der Nominatim API in Python und PostgreSQL/PostGIS.

Diskussion:

- Kunde 2 aus Bellevue ist mit einer Distanz von 438 Metern am nächsten an einer Tischtennistisch.
- Kunde 1 aus der Vulkanstrasse liegt mit 693 Metern an zweiter Stelle.
- Kunde 3 (der Weihnachtsmann 🎅) taucht nicht in den Ergebnissen auf, da die Geokodierung keine Adresse für diesen Ort gefunden hat.

3. Fazit und Ausblick

Es wurden hauptsächlich die folgende Software verwendet: der Overpass-API-Dienst und das PostgreSQL/PostGIS-Tool (siehe Abb. 4).

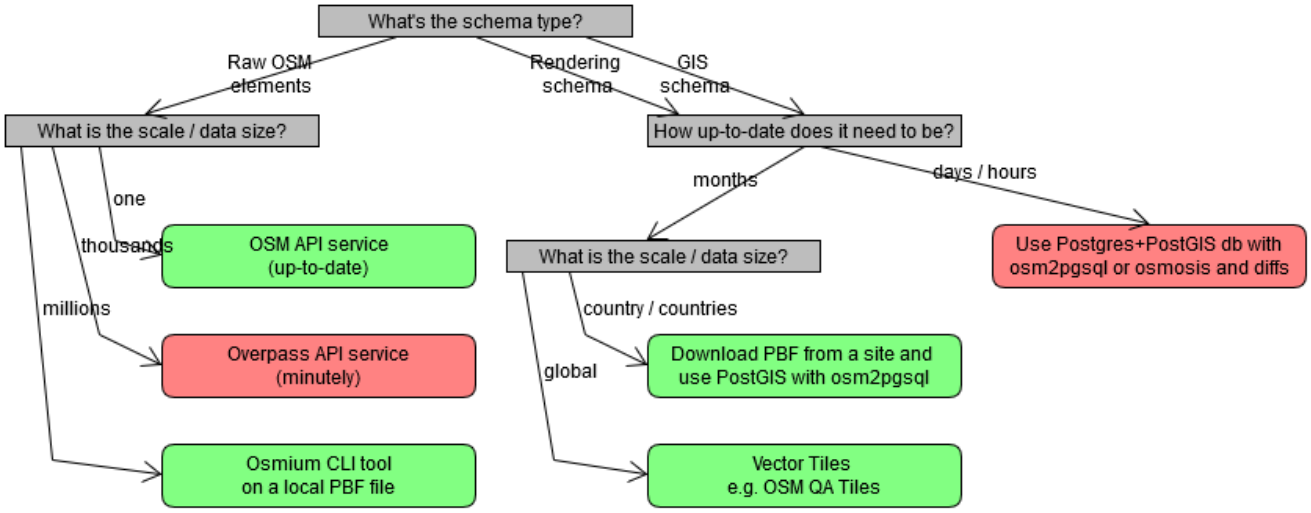

Figure 4. Entscheidungsbaum für Werkzeuge zum Zugriff auf OSM-Daten. Hinweis: Die rot markierten Blöcke sind die in diesem Dokument verwendeten Dienste und Werkzeuge. (Quelle: Inspiriert von GIScience News Blog, 10. September 2020 by Moritz Schott)

Die Vorteile des Overpass-API-Dienstes sind, dass es sich um einen Webdienst handelt und stets aktuelle Daten enthält. Die Nachteile sind, dass die Ergebnisse nur eine begrenzte Anzahl von Datensätzen enthalten können und die Abfragesprache relativ kompliziert und eingeschränkt ist.

Der Vorteil von PostgreSQL/PostGIS liegt in der leistungsfähigen räumlichen SQL-Abfragefunktionalität (siehe z.B. diese Tipps und Tricks). Die Nachteile sind der Installationsaufwand sowie die benötigten lokalen Ressourcen – zusätzlich dazu, dass PostGIS-Abfragen (und der osm2pgsql-Bulk-Loader) derzeit noch nicht parallelisiert werden können. Dennoch sind sie für viele Anwendungen schnell genug, wenn sie mit den richtigen Indizes und Operatoren (z.B. KNN <->) effizient genutzt werden.

Neben dem speziellen Webdienst Overpass API, der mit Python genutzt wird, gibt es auch andere Ansätze, wie z.B. SQL in Kombination mit pgRouting, einer weiteren PostgreSQL-Erweiterung ähnlich PostGIS.

OSMnx ist äusserst nützlich und der Kurs deckt viele GIS-Themen ab, von Vektor-Overlay- und Netzwerk-Analysen bis hin zur Rasterdatenanalyse.

Letztendlich dreht sich alles um Daten. Die entscheidende Frage lautet daher: Wie lassen sich weitere offene Geodaten finden? Diese Frage hat keine einfache Antwort und erfordert viel Erfahrung. Eine gute gute Quelle offener Geodaten sind Portale wie opendata.swiss oder “Freie Geodaten”.

Abschliessend sollte noch einmal das Lernportal OpenSchoolMaps erwähnt werden, das regelmässig mit neuen Lernmaterialien aktualisiert wird.

4. 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
- Ein interessanter Kurs behandelt ausserdem das Thema "Herunterladen und Visualisieren von OpenStreetMap-Daten mit OSMnx" (Python)
- "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://automating-gis-processes.github.io/site/notebooks/L6/retrieve_osm_data.html
- Weitere Python Libraries:
  - osm2geojson: https://github.com/mvexel/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://towardsdatascience.com/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? Wende dich an OpenStreetMap Schweiz oder Stefan Keller!