# Crowdsourcing-Daten nutzen

November 3, 2021

## 1 Crowdsouring-Daten nutzen

Daten aus OpenStreetMap (OSM) und WikiData zusammen mit städtischen Daten nutzen

**Jupyter-Notebook laden: bit.ly/gisforum2021**

### 1.1 Initialisierung

[ ]:
```
# Python Pakete installieren (auf JupyterHub nicht nötig!)
# pip install -r requirements.txt
```

[ ]:
```
import os
import json
from pprint import pprint

import requests
import folium
import geopandas
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import matplotlib.dates as mdates
from IPython.display import HTML, JSON, display, display_javascript

import utils
```

[ ]:
```
# Inhalt von utils anschauen
#%pycat utils/__init__.py
```

[ ]:
```
lv95 = 'EPSG:2056'
wgs84 = 'EPSG:4326'
```

## 2 Städtische Daten «Kunst im öffentlichen Raum (KiöR)»

Im ersten Teil schauen wir uns an, wie wir städtische Daten mit Daten aus OpenStreetMap (OSM) anreichern können. Dazu laden wir uns zuerst den Datensatz «Kunst im öffentlichen Raum» via WFS.

### 2.1 Daten finden

Auf dem Open-Data-Katalog finden wir den Datensatz Kunst im Stadtraum. Wenn wir den Datensatz als GeoJSON herunterladen wollen, werden wir entsprechend zum Geoportal weitergeleitet.

[ ]:
```
# Variablen setzen
kioer_geojson_url = 'https://www.ogd.stadt-zuerich.ch/wfs/geoportal/
 ↪Kunst_im_Stadtraum?service=WFS&version=1.1.
 ↪0&request=GetFeature&outputFormat=GeoJSON&typename=view_kioer'
kioer_layer = 'view_kioer'
wfs_url = 'https://www.ogd.stadt-zuerich.ch/wfs/geoportal/Kunst_im_Stadtraum'
layers = utils.get_layers_from_wfs(wfs_url) # GetCapabilitites
layers
```

### 2.2 WFS GetFeature Request absetzen

[ ]:
```
r = requests.get(wfs_url, params={
    'service': 'WFS',
    'version': '1.0.0',
    'request': 'GetFeature',
    'typename': kioer_layer,
    'outputFormat': 'GeoJSON'
})

kioer_geo = r.json()
display(kioer_geo)
```

## 3 Daten aus OpenStreetMap laden

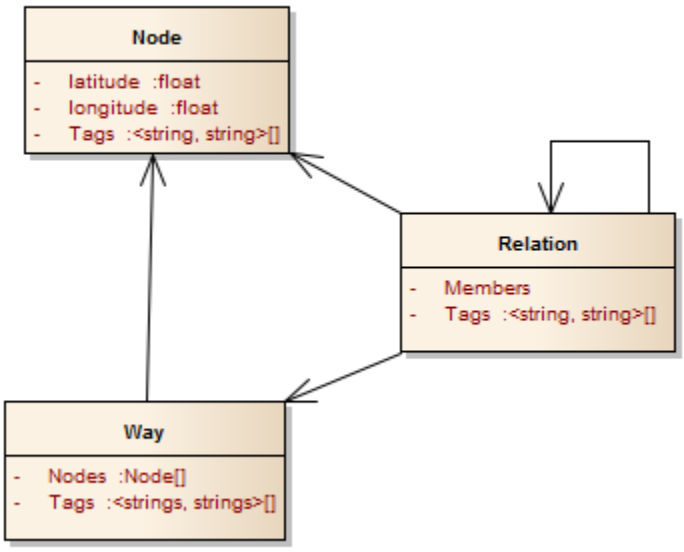
Wir suchen thematisch passende Daten aus OpenStreetMap. Um diese zu finden, lohnt es sich das OpenStreetMap Wiki zu durchsuchen und passende Tags zu finden.

**ACHTUNG:** Daten aus OpenStreetMap stehen unter der Open Data Commons Open Database License (ODbL). D.h. die Verwendung der Daten ist erlaubt wenn:

- die Quelle erwähnt wird und
- allfällige Änderungen unter derselben Lizenz zur Verfügung stehen

### 3.1 OSM-Daten laden

Daten von OpenStreetMap (OSM) können u.a. via Overpass API geladen werden. Overpass hat eine eigene Abfragesprache (Overpass QL), mit der Objekte (Nodes, Ways, Relations) abgefragt werden können.




Bildquelle: © Itinero

### 3.2 Overpass Query absetzen

Query in Overpass Turbo ausführen

[ ]:
```
artwork_zh = """
/*
Alle Kunstwerke (tourism=artwork) in der Stadt Zürich
*/
[out:json];
area["name"="Zürich"]["wikipedia"="de:Zürich"]->.perimeter;
(
  nwr[tourism=artwork](area.perimeter);
);
out center;
"""
osm_result = utils.overpass_query(artwork_zh)
osm_result
```

### 3.3 Karte mit den Resultaten

[ ]:
```
# Basiskarte
m = utils.base_map()

# KiöR-Daten hinzufügen
kioer_layer = folium.FeatureGroup(name='KiöR', show=True)
utils.style_layer(kioer_geo, kioer_layer, icon_color='#031cff',␣
 ↪icon='certificate', prefix='fa')
kioer_layer.add_to(m)
```

```
# OSM-Daten hinzufügen
osm_layer = folium.FeatureGroup(name='OSM: tourism=artwork', show=True)
utils.style_layer(osm_result, osm_layer, icon_color='#ff0000', icon='fire',␣
↪prefix='fa')
osm_layer.add_to(m)

# Add controls for layers
folium.LayerControl().add_to(m)
m
```

## 3.4 Spatial Join der zwei Quellen

Um die beiden Quellen zu matchen gibt es grundsätzlich zwei Möglichkeiten:

1. Join über Attribute (z.B. Titel des Kunstwerks)
2. Räumlicher Join (Objekte die nah beieinander sind)

Oder eine Kombination der beiden Ansätze.

Im folgenden schauen wir uns den **Räumlichen Join («Spatial Join»)** genauer an.

### 3.4.1 GeoDataFrame für geopandas erstellen

[ ]:
```
kioer_df = geopandas.GeoDataFrame.from_features(kioer_geo, crs=wgs84)
kioer_df.head()
```

[ ]:
```
osm_df = geopandas.GeoDataFrame.from_features(osm_result, crs=wgs84)
osm_df.head()
```

### 3.4.2 Geometrien für Join vorbereiten

[ ]:
```
# Einträge mit leerer Geometrie
kioer_df[(pd.isna(kioer_df.geometry))].head()
```

[ ]:
```
# Alle Einträge mit leerer Geometrie entfernen
kior_buf = kioer_df.dropna(subset=['geometry']).reset_index(drop=True)
osm_buf = osm_df.dropna(subset=['geometry']).reset_index(drop=True)

# CRS zu LV95 re-projezieren
kior_buf = kior_buf.to_crs(lv95) # Konvertieren zu 2D Koordinatensystem
osm_buf = osm_buf.to_crs(lv95) # Konvertieren zu 2D Koordinatensystem

# Buffer um die Geometrien hinzufügen (10 Meter)
kior_buf['geometry'] = kior_buf.geometry.buffer(10)
osm_buf['geometry'] = osm_buf.geometry.buffer(10)
```

### 3.4.3 Vorschau der Geometrien auf der Karte

[ ]:
```
# Basiskarte
bm = utils.base_map()

folium.features.GeoJson(
    kior_buf.to_crs(wgs84).to_json(),
    tooltip=folium.features.GeoJsonTooltip(
        fields=['titel', 'autoren', 'datierung', 'material'],
        aliases=['Titel:', 'Künstler:', 'Datierung:', 'Material:'],
    )
).add_to(bm)

folium.features.GeoJson(
    osm_buf.to_crs(wgs84).to_json(),
    style_function=lambda x: {'fillColor': '#FF0000', 'color': '#FF0000'},
    tooltip=folium.features.GeoJsonTooltip(
        fields=['name', 'artist_name', 'wikidata'],
        aliases=['Titel:', 'Künstler:', "Wikidata:"],
    )
).add_to(bm)
bm
```

### 3.4.4 Join durchführen

[ ]:
```
# spatial join über die beiden Geometrien
sjoin_kunst = geopandas.sjoin(kior_buf, osm_buf, how='left',␣
 ↪predicate='intersects', lsuffix='kior', rsuffix='osm').reset_index()

# Wie zurück zu WGS84
sjoin_kunst = sjoin_kunst.to_crs(wgs84)


with pd.option_context('display.max_rows', None, 'display.max_columns', None):
    display(sjoin_kunst[['titel', 'name', 'autoren', 'artist_name',␣
 ↪'material_kior', 'material_osm', 'datierung']])
```

### 3.4.5 Resultat des Joins auf der Karte

[ ]:
```
folium.features.GeoJson(
    sjoin_kunst.to_json(),
    style_function=lambda x: {'fillColor': '#09bd63', 'color': '#09bd63'},
    tooltip=folium.features.GeoJsonTooltip(
        fields=['name', 'artist_name', 'wikidata'],
        aliases=['Titel:', 'Künstler:', "Wikidata:"],
    )
).add_to(bm)
```

```
bm
```

### 3.4.6 OSM-Einträge ohne Match

[ ]:
```
# OSM-Einträge ohne Match
osm_no_match = osm_df[(~osm_df.id.isin(sjoin_kunst.id))].reset_index()
osm_no_match.head()
```

### 3.4.7 Geopackage erstellen

Das Result lässt sich sehr einfach als Geopackage exportieren.

Im Beispiel ein Layer für die gematchten und ein Layer für die nicht-gematchten Einträge.

**Das Geopackage lässt sich anschliessend z.B. in QGIS weiterverarbeiten**

[ ]:
```
osm_no_match.to_file(os.path.join('..', 'kunst_package.gpkg'),␣
↪layer='osm_no_match', driver="GPKG")
sjoin_kunst.to_file(os.path.join('..', 'kunst_package.gpkg'),␣
↪layer='kioer_osm_match', driver="GPKG")
```

# 4 Daten aus WikiData laden

WikiData ist ein Schwesterprojekt von Wikipedia. Es beinhaltet strukturierte Daten, welche via API abgefragt werden können (REST oder SPARQL).

Alle Daten aus WikiData sind CreativeCommons-Zero (CC0) lizenziert, können also bedenkenlos weiterverwendet werden.

## 4.1 Daten via WikiData-Verweise von OSM holen

OpenStreetMap Node «Heureka»

[ ]:
```
# WikiData Verweise von OpenStreetMap
osm_kunst_wd = osm_df[(~pd.isna(osm_df.wikidata))] # alle Einträge, bei denen␣
↪"wikidata" nicht leer ist
osm_kunst_wd = osm_kunst_wd[['geometry', 'type', 'id', 'artist_name', 'artist:
↪wikidata', 'name', 'wikidata']].reset_index(drop=True)
osm_kunst_wd.head()
```

## 4.2 Beispiel-Abfrage nach einem WikiData Item

Wikidata-Item «Heureka»

[ ]:
```
wikidata_item = utils.wikidata_item(osm_kunst_wd['wikidata'].iloc[0])
pprint(wikidata_item['sitelinks'])
```

**Bilder aus Wikimedia Commons laden:**

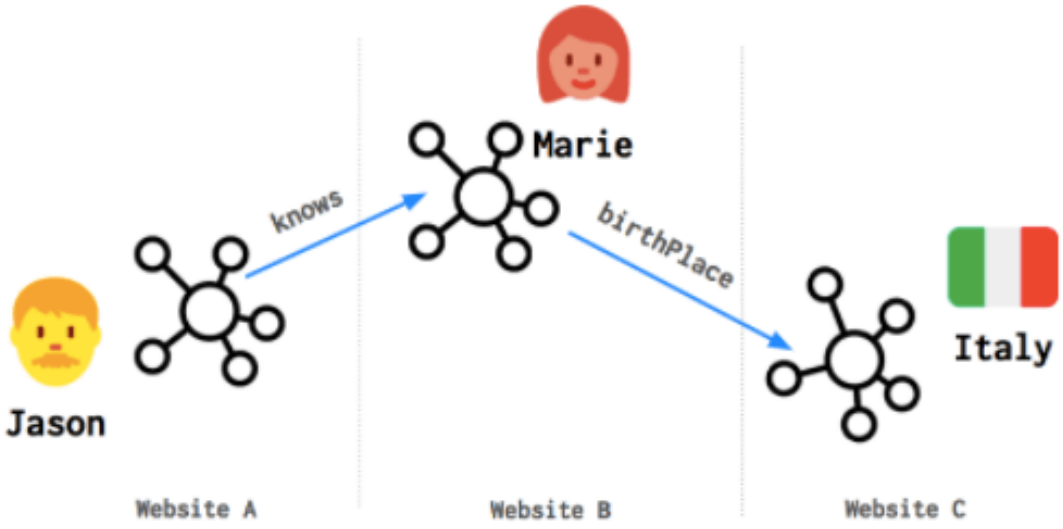
[ ]:
```
# Bilder laden
category = wikidata_item['sitelinks']['commonswiki']['title']
urls = utils.images_from_commons_category(category, image_size=500)
urls
```

[ ]:
```
#Bilder anzeigen
display(HTML(''.join([utils.img_html(url) for url in urls])))
```

### 4.3 Daten via SPARQL aus WikiData beziehen

Um Daten aus WikiData zu laden, können Abfragen mit SPARQL gemacht werden. SPARQL ist eine SQL-ähnliche Abfragesprache für Linked Data.

Die Idee von Linked Data ist es, einen Informations-Graphen in Form von sogenannten «Triples» abzubilden:




**Triple = Subjekt (z.B. Marie) -> Prädikat (z.B. birthPlace) -> Objekt (z.B. Italy)**

Bildquelle: © WordLift

#### 4.3.1 WikiData SPARQL-Query absetzen

Link zum Wikidata Query Service

[ ]:
```
# direkte SPARQL-Query auf WikiData
wd_kunstwerke_query = """
SELECT DISTINCT ?artwork ?artworkLabel ?creator ?creatorLabel ?creatorBirthday ?
 ↪createDateOfDeath ?geo
WHERE
{
  ?artwork wdt:P136 wd:Q557141 .           # Genre "Kunst im öffentlichen Raum"
  ?artwork wdt:P131 wd:Q72 .               # liegt in Zürich
```

```
  OPTIONAL {
    ?artwork wdt:P170 ?creator .           # Urheber des Werks
    ?creator wdt:P569 ?creatorBirthday .   # Geburtsdatum des Urhebers
    ?creator wdt:P570 ?createDateOfDeath . # Todesdatum des Urhebers
  }
  OPTIONAL {
    ?artwork wdt:P625 ?geo .               # Koordinaten für Kartenansicht
  }

  SERVICE wikibase:label {
    bd:serviceParam wikibase:language "[AUTO_LANGUAGE],de,en"
  }
}
ORDER BY ?artworkLabel
"""
wd_result = utils.wikidata_query(wd_kunstwerke_query)
wikidata_df = pd.DataFrame(wd_result)
wikidata_df.head()
```

### 4.3.2 WikiData-Resultat als GeoDataFrame

[ ]:
```
wikidata_df[(pd.isna(wikidata_df.geo))]
```

[ ]:
```
# remove entries with empty coordinates
wikidata_df = wikidata_df.dropna(subset=['geo']).reset_index(drop=True)
wikidata_df.head()
```

[ ]:
```
# Erstelle GeoDataFrame
#wikidata_gdf = geopandas.GeoDataFrame(wikidata_df, geometry=geopandas.
 ↪points_from_xy(wikidata_df.lon, wikidata_df.lat))
wikidata_gdf = geopandas.GeoDataFrame(wikidata_df, geometry=geopandas.GeoSeries.
 ↪from_wkt(wikidata_df.geo))
wikidata_gdf = wikidata_gdf.drop(columns=['geo'])
wikidata_gdf = wikidata_gdf.set_crs(wgs84)
wikidata_gdf = wikidata_gdf.dropna(subset=['geometry']).reset_index(drop=True)
wikidata_gdf.head()
```

### 4.3.3 WikiData-Items auf Karte darstellen

[ ]:
```
# Basiskarte
wikimap = utils.base_map()

wd_layer = folium.FeatureGroup(name='WikiData: public art', show=True)
utils.style_layer(wikidata_gdf, wd_layer, color='green', icon='eye',␣
 ↪prefix='fa')
wd_layer.add_to(wikimap)
```

```
wikimap
```

# 5 Food for thought / Aufgaben für Fortgeschrittene

1. Wie können wir einen Spatial Join zwischen dem KiöR und dem WikiData Dataset machen?
2. Wie kann eine Kombination aus Spatial Join und Attribute Join gemacht werden (um die Qualität der Matches zu verbessern)?
3. Wie könnten Daten aus OSM und WikiData in städtische Daten integriert werden?
   - Als Datenquelle?
   - Als Plausibilitätscheck?
   - Daten abgleichen?
   - …?
4. Wie könnten städtische Daten in OSM und WikiData integriert werden?
   - via OGD?
   - direkt via Bot?
   - …?

# 6 Exkurs: Daten von geo.admin.ch via SPARQL beziehen

Nachdem wir oben gesehen haben, dass wir (Geo-)Daten via SPARQL beziehen können, gibt der folgende Exkurs einen kleinen Einblick in die Geodaten von geo.admin.ch, welche als Linked Data zur Verfügung gestellt werden.

Diese umfassen u.a.:

- swissBOUNDARIES 3D
- Haltestellen des öffentlichen Verkehrs
- Generalisierte Grenzen G1
- Strassenverkehrsunfallorte

Wir wollen nun am Beispiel von swissBOUNDARIES zeigen, wie wir eine (stets aktuelle) Liste von Schweizer Gemeinden beziehen können.

## 6.1 SPARQL Query für CH-Gemeinden

SPARQL Query auf ld.geo.admin.ch ausführen

[ ]:
```
gemeinde_query = """
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX schema: <http://schema.org/>
PREFIX gn: <http://www.geonames.org/ontology#>
PREFIX ch: <https://ld.geo.admin.ch/def/>
PREFIX dct: <http://purl.org/dc/terms/>
PREFIX geosparql: <http://www.opengis.net/ont/geosparql#>

SELECT ?bfsNummer ?GemeindeName ?RegionName ?KantonName ?Bevoelkerung ?geoWKT
```

```
WHERE
{
  ?AdminArea rdf:type schema:AdministrativeArea .
  ?AdminArea dct:hasVersion ?Gemeinde .
  ?Gemeinde gn:featureCode gn:A.ADM3 .
  ?Gemeinde schema:name ?GemeindeName .
  ?Gemeinde ch:bfsNumber ?bfsNummer .
  ?Gemeinde gn:parentADM1 ?Kanton .
  ?Kanton schema:name ?KantonName .
  ?Gemeinde dct:issued ?Datum .
  ?Gemeinde schema:validUntil ?ValidDatum .
  ?Gemeinde gn:population ?Bevoelkerung .
  ?Gemeinde geosparql:hasGeometry/geosparql:asWKT ?geoWKT .

  OPTIONAL {
    ?Gemeinde gn:parentADM2 ?Region .
    ?Region schema:name ?RegionName .
  }

  # Filter auf die aktuellste, gültige Version
  FILTER (?ValidDatum >= xsd:date(NOW()))
  FILTER NOT EXISTS {
      ?AdminArea dct:hasVersion/dct:issued ?AusstellDatum
      filter (?AusstellDatum > ?Datum)
  }
}
ORDER BY ASC(?GemeindeName)
LIMIT 3000
"""
gde_result = utils.geoadmin_query(gemeinde_query)
gde_df = pd.DataFrame(gde_result)
gde_df.bfsNummer = pd.to_numeric(gde_df.bfsNummer)
gde_df.Bevoelkerung = pd.to_numeric(gde_df.Bevoelkerung)
gde_df.head()
```

### 6.2 Gemeinden als GeoDataFrame

[ ]:
```
gde_gdf = geopandas.GeoDataFrame(gde_df, geometry=geopandas.GeoSeries.
 ↪from_wkt(gde_df.geoWKT))
gde_gdf = gde_gdf.drop(columns=['geoWKT'])
gde_gdf = gde_gdf.set_crs(wgs84)
gde_gdf = gde_gdf.dropna(subset=['geometry']).reset_index(drop=True)
gde_gdf.head()
```

### 6.3 Visualisierungs-Beispiele

Im folgenden noch ein paar Beispiele, wie diese Daten visualisiert werden können.

### 6.3.1 Gemeindegrenzen auf Karte darstellen

[ ]:
```
gemeinde_map = utils.base_map(location=[46.25, 8.53], zoom=8)
folium.features.GeoJson(
    gde_gdf.to_json(),
    style_function=lambda x: {'fillColor': '#17a0c2', 'color': '#17a0c2'},
    tooltip=folium.features.GeoJsonTooltip(
        fields=['GemeindeName', 'Bevoelkerung', 'bfsNummer'],
        aliases=['Gemeinde:', 'Bevölkerung:', "BFS Nummer:"],
    )
).add_to(gemeinde_map)
gemeinde_map
```

### 6.3.2 10 grösste Städte

[ ]:
```
utils.use_style('fivethirtyeight')
fig, ax = plt.subplots()

top10 = gde_gdf.nlargest(10, ['Bevoelkerung'])
top10.plot(kind='barh', y='Bevoelkerung', x="GemeindeName",␣
 ↪label="Bevölkerung", ax=ax)
ax.legend().set_visible(False)
ax.set_ylabel('Bevölkerung')
ax.set_xlabel('Gemeinde')
ax.invert_yaxis()
ax.xaxis.set_major_formatter(ticker.EngFormatter())
plt.show()
```

### 6.3.3 Bevölkerungswachstum in Zürich

Um das Bevölkerungswachstum zu visualisieren, holen wir zuerst die Bevölkerungszahlen (gemäss BFS-Definition) von Zürich.

[ ]:
```
bev_zurich_query = """
PREFIX schema: <http://schema.org/>
PREFIX gn: <http://www.geonames.org/ontology#>
PREFIX dct: <http://purl.org/dc/terms/>

SELECT ?GemeindeName ?ValidDatum ?Bevoelkerung
WHERE
{
  VALUES ?AdminArea { <https://ld.geo.admin.ch/boundaries/municipality/261> } #␣
 ↪Zürich
  ?AdminArea dct:hasVersion ?Gemeinde .
  ?Gemeinde schema:name ?GemeindeName .
  ?Gemeinde schema:validUntil ?ValidDatum .
  ?Gemeinde gn:population ?Bevoelkerung .
```

```
}
ORDER BY ASC(?ValidDatum)
LIMIT 100
"""
zh_result = utils.geoadmin_query(bev_zurich_query)
zh_df = pd.DataFrame(zh_result)
zh_df.Bevoelkerung = pd.to_numeric(zh_df.Bevoelkerung)
zh_df.ValidDatum = pd.to_datetime(zh_df.ValidDatum, format='%Y-%m-%d')
zh_df
```

[ ]:
```
utils.use_style('fivethirtyeight')
fig, ax = plt.subplots()

zh_df.plot(kind='line', y='Bevoelkerung', x="ValidDatum", label="Bevölkerung",␣
↪ax=ax)
ax.legend().set_visible(False)
ax.set_ylim(bottom=0, top=450_000) # immer bei 0 starten
ax.set_ylabel('Bevölkerung')
ax.set_xlabel('Jahr')
plt.setp(ax.get_yticklabels()[0], visible=False)
plt.show()
```

# 7 Anhang und Tutorials

## 7.1 OpenStreetMap (OSM)

- OpenStreetMap Wiki
  - tourism=artwork
  - Taginfo tourism=artwork
- Overpass API: Abfragen für OpenStreetMap
  - Overpass Turbo Weltweit
  - Overpass Turbo Schweiz
- OpenSchoolMaps: Unterrichtsmaterial inkl. Einsteigerhilfe zu OpenStreetMap
  - OpenStreetMap-Daten beziehen und mit QGIS 3 nutzen (Overpass, QuickOSM-Plugin)
  - Raumanalyse Vektordaten - Folgen-Abschätzung eines Autobahn-Baus mit QGIS
- learnOSM: Schritt-für-Schritt-Anleitungen

## 7.2 WikiData

- WikiData Query Service: Tool für SPARQL-Abfragen inkl. vielen Beispielen
- WikiData Training: Einführung in WikiData
- WikiData SPARQL Tutorial
- Wikidata Jupyter Notebooks von OpenDataZurich: MediaWiki API, SPARQL, Bots
- EUCLID: Querying Linked Data: SPARQL-Kurs

[ ]: