



Utrecht University

Applied Data Science Master's degree programme

Spatial Data Analysis and Simulation Modelling

Instruction manual for Lab 1.2: **geodata quality**

Document version: 1.0

Document modified: 2020.10.13

Dr. Simon Scheider

Department of Human Geography and Spatial Planning

Faculty of Geosciences

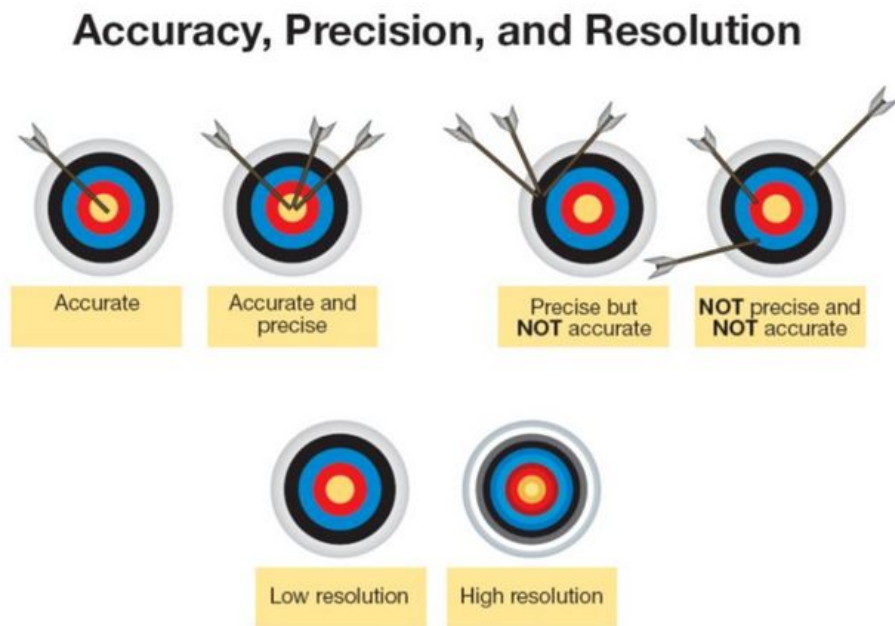
s.scheider@uu.nl

Table of contents

Assessing the quality of geodata	3
Geodata of different quality about schools in Amsterdam	6
Load openTopo background map (WMTS) from PDOK and zoom to Amsterdam	6
Load official school data using geojson API	7
Access school data from Open Street Map	8
Save the datasets as shapefiles, and project them into "RD new"	11
Make a single point layer out of the two OSM layers	12
Determine the extent of a layer	13
Determine the resolution of a (polygon vector) layer	15
Measuring completeness and spatial accuracy	17
Matching schools based on distance and similarity of names	18
Assess completeness and spatial accuracy	21
Assignment: assess quality of other geodata sources, and document their quality	22

Assessing the quality of geodata

Before you can make any valid use of a geodata source for analysis or modeling purposes, it is important that you are aware of its quality. Geodata of unsuitable quality will result in invalid or questionable modeling results. While the dimensions of geodata quality form part of its **metadata**¹ and resemble ordinary quality dimensions of any data, they are also special in many respects. First, geodata comes with spatial quality dimensions, accounting for the geometric component and the coordinate reference systems (CRS) of the data. Furthermore, geodata comes in layers (datasets of homogeneous geometry and phenomena types).



In this lab, you will learn the principles of and how to assess the following geodata quality dimensions, with a focus on Vector data quality:

- **Spatial accuracy:** means the degree of correspondence of the location of an entity with the location in a reference data set that is considered “ground” truth. Spatial accuracy is assessed in terms of a **location error**, measured in terms of a distance of geometries (point, line, region) of some entity to the corresponding geometries in the reference data set. Both mean and variance of this error (variance is sometimes also called **precision**) is relevant (see Fig. above). For example, we might be interested in the average location error of places (Points of Interest) recorded in Open Street Map (OSM).
- **Spatial resolution:** means the degree of detail recorded in the geometry of a layer. Similar to precision in digital number systems (e.g. floating point numbers), where a given number covers a range of real numbers not further distinguishable on the computer, spatial resolution measures the smallest distinguishable size within a layer. For image data, this corresponds to the pixel size. For vector data, we can measure resolution only for line and region (polygon) data, based on measuring the length of the smallest geometric line segment that makes up any geometry of that

¹ <https://www.w3.org/TR/sdw-bp/#bp-metadata>

layer (see Fig. 1 below). For example, a coastline can be measured up to varying degrees of detail, corresponding to different lengths of underlying segments (see Fig. 2 below). Resolution of data is relevant to the **map scale level** of a cartographic map, which is the range of map scales (e.g. 1:50.000-1:1.000.000) in which a map can be meaningfully displayed on a computer screen. This depends on its spatial resolution. Furthermore, layers on very different resolution levels should never be combined with each other.

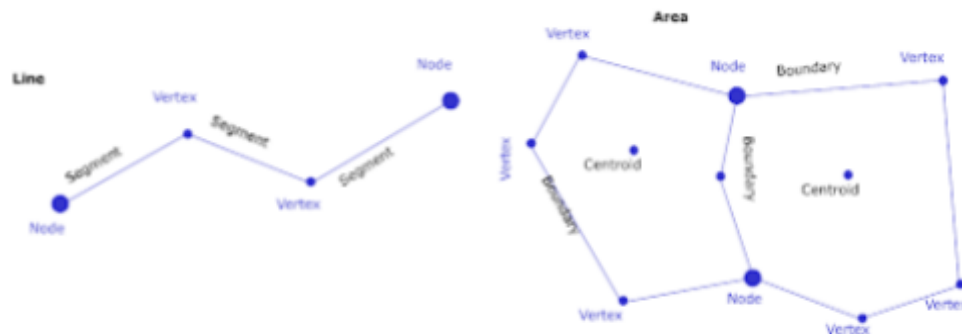


Fig 1: Composition of lines and regions from line segments and boundary segments.



Fig. 2: The coastline paradox: Great Britain in different resolutions, leaving open the question of its precise circumference.

- **Spatial extent** (completeness I): Geodata of high resolution often covers only a local, defined portion of the Earth's surface. It is important to know the extent of a layer, because we can only combine different data sources if their extent at least overlaps. For example, a layer of housing in Amsterdam cannot be combined with a layer of Utrecht's roads. The spatial extent is usually given in terms of a **minimum bounding box** (MBB), a rectangular geometry given by minimum and maximum coordinates

that enclose the geometries of the layer.

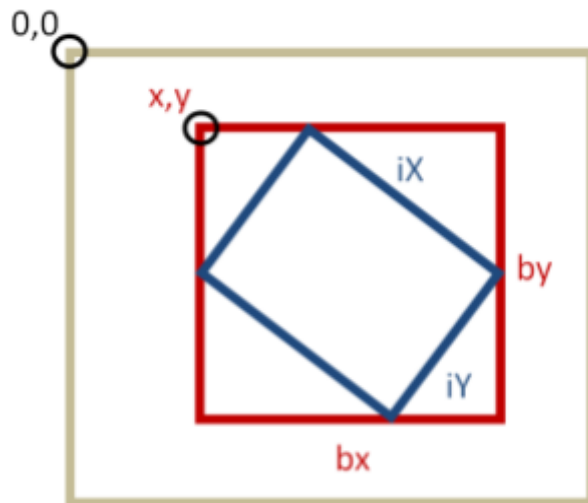


Fig 3: Minimum bounding rectangle (red) of a geometry (blue), consisting of minimum and maximum coordinates in a CRS.


- **Thematic completeness** (completeness II): This corresponds to the completeness of phenomena recorded within the extent of a layer, measured with respect to a reference data set. For example, we would like to know whether, within the spatial extent of a layer, all houses that are known to exist have actually been recorded.

Besides **spatial** accuracy, resolution and completeness, which refers to the geometry component, we also may have to take into account **attribute** accuracy, resolution and completeness. This is because geodata is always a combination of geometric with ordinary measurements. Furthermore, there are more quality dimensions (see lecture). However, in this lab, we are focusing on the spatial quality dimensions explained above.

Geodata of different quality about schools in Amsterdam

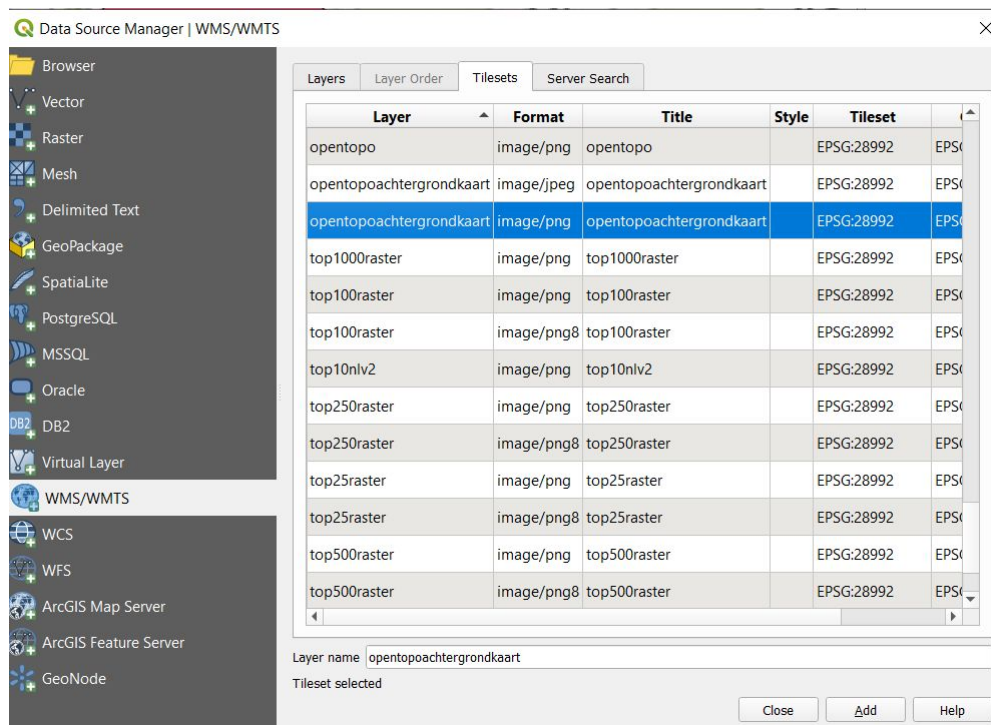
Suppose our goal is to assess the quality and coverage of Amsterdam's school infrastructure. We start by getting some geodata about school locations from diverse sources and bring them all into the common Dutch reference system "RD new".

Load openTopo background map (WMTS) from PDOK and zoom to Amsterdam

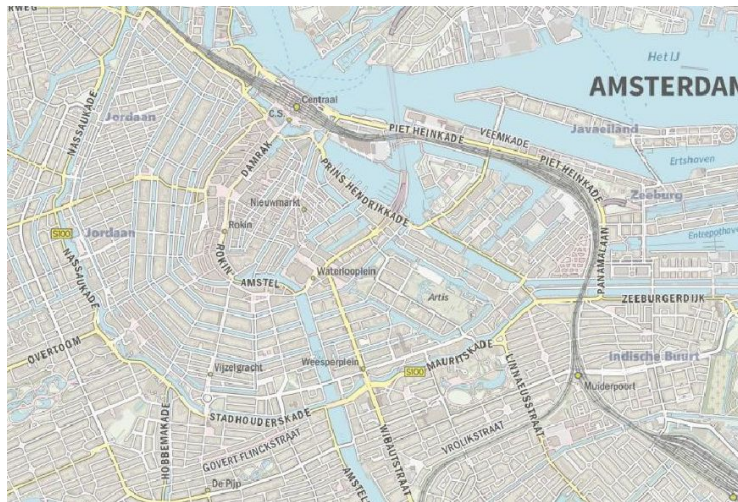
Open the Data Source Manager () of QGIS. Click on WMS/WMTS, create a "New" service connection and enter the following URL:

<https://geodata.nationaalgeoregister.nl/tiles/service/wmts?request=GetCapabilities&service=WMTS>

Then click "Connect" ("Ignore" any possible SSL error), wait till QGIS connects to the service and then open Tilesets (see Fig. below). Open the background layer "opentopoachtergrondkaart" by selecting it and pressing "Add". Select the default transformation method.



Now a topographic background map in RD New projection should appear in the map window. Zoom the map to Amsterdam, by pointing with your mouse into the map window and by turning the mouse wheel:

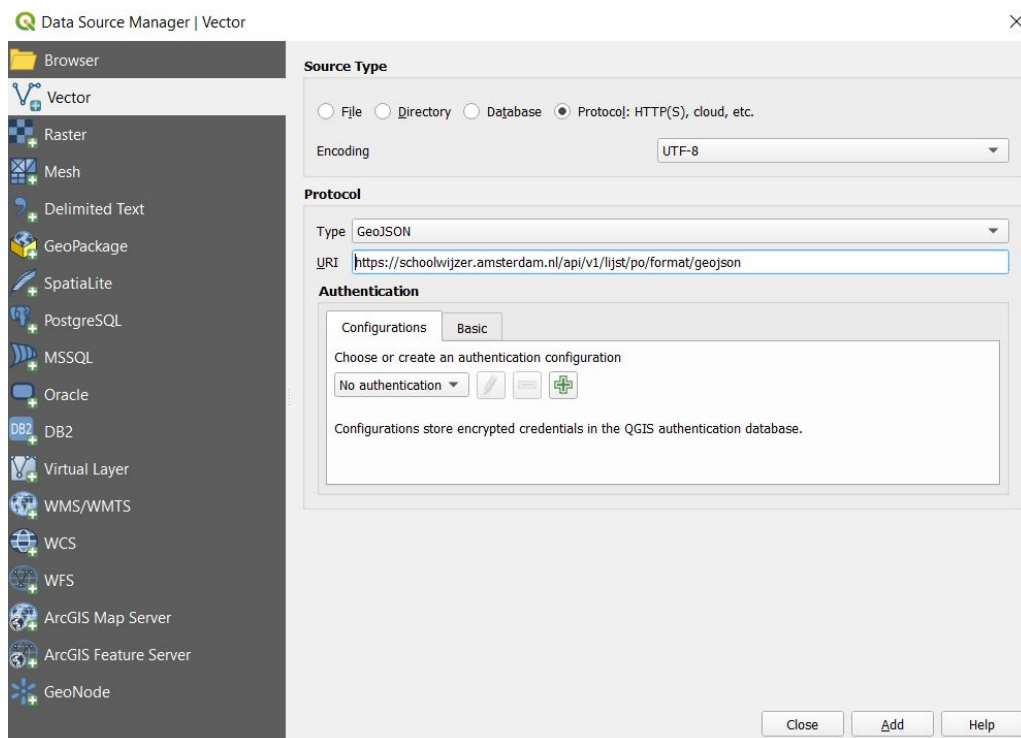


Load official school data using geojson API

The site (<https://schoolwijzer.amsterdam.nl/nl/api/documentatie>) contains official data about all schools in Amsterdam. Take a look at the API documentation. Make sure you understand that this is a RESTful API where the URI snippet “po” means primary schools, while “vo” means secondary schools. Furthermore, the URI snippet “format/geojson” will return the GeoJSON² format, which is a standard format for exchanging geodata as JSON.

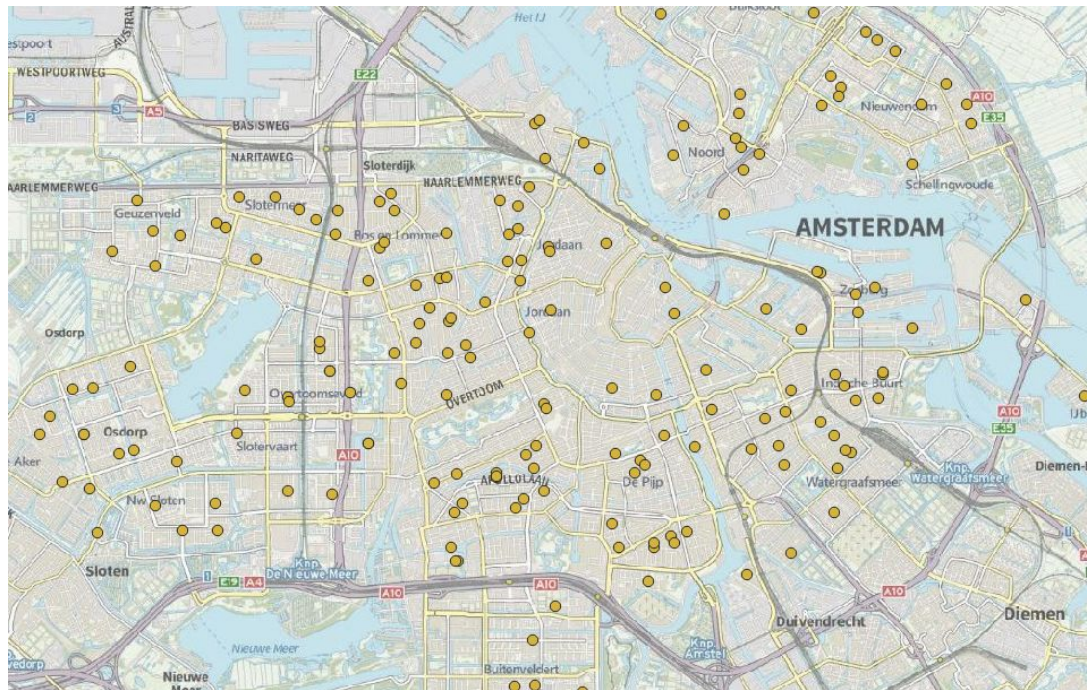
You can either open the Data Source manager again, click on Vector and choose HTTP protocol GeoJSON (see Fig. below), or (in case service does not work properly) just enter the following URI into a browser window, save and load the data as a .geojson file with the Vector/File option:

<https://schoolwijzer.amsterdam.nl/api/v1/lijst/po/format/geojson>



² <https://geojson.org/>


As a result, you should see the following layer in the map window:



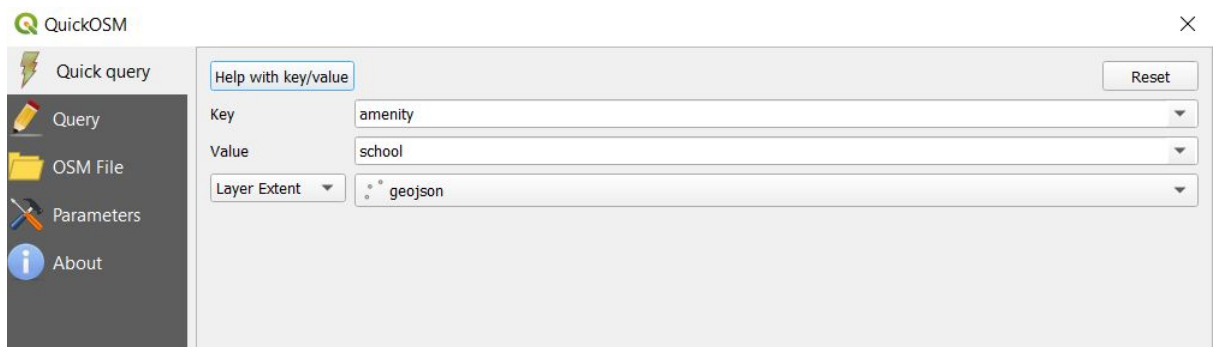
Access school data from Open Street Map

While schoolwijzer data is an official (and maybe more trustworthy?) geodata source, we would like to compare its quality with a volunteered geographic information (VGI) source, such as Open Street Map (OSM).

Loading OSM can be done using e.g. the **QuickOSM** plugin. If you have not already installed this plugin, you can easily do so via the Plugin manager in the QGIS menu, after

which the QuickOSM icon should appear in your menu (). Open QuickOSM.

OSM data is organized and can be retrieved via **key-value pairs**. Enter the pair **amenity/school** . Note that OSM data does not have a defined extent, as it is (potentially) global. Therefore, you need to provide a layer extent used to cut out OSM data. IN our case, we just use the one of the given schooldata in geojson:



Then open the “Query tab” and press “Generate query”, where the corresponding Overpass³ query appears. Open the “Advanced” menu and select “Points” and “Multipolygons” as output geometry types. Then download multipolygons and points by pressing “Run query”:

The screenshot shows the QuickOSM web interface. On the left is a sidebar with navigation links: "Quick query", "Query", "OSM File", "Parameters", and "About". The main area is titled "Overpass query" and contains a text editor with the following XML query:

```
<osm-script output="xml" timeout="25">
  <union>
    <query type="node">
      <has-kv k="amenity" v="school"/>
      <bbox-query {{bbox}}/>
    </query>
    <query type="way">
      <has-kv k="amenity" v="school"/>
      <bbox-query {{bbox}}/>
    </query>
    <query type="relation">
      <has-kv k="amenity" v="school"/>
      <bbox-query {{bbox}}/>
    </query>
  </union>
  <union>
    <item/>
    <recurse type="down"/>
  </union>
  <print mode="body"/>
</osm-script>
```

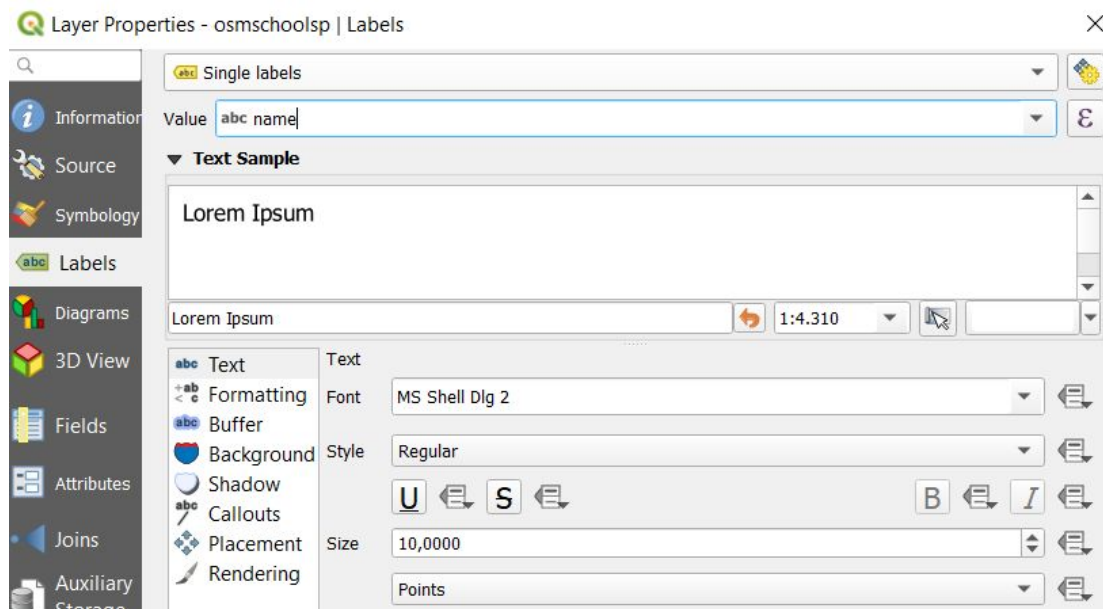
Below the query editor is an "Advanced" section with various settings:

- Geocode area: Can be overridden
- Bbox or center: Canvas Extent, * * basisscholen
- Outputs:
 - ☒ Points: col1,col2,col3
 - ☐ Lines: or let empty
 - ☐ Multilinestrings
 - ☒ Multipolygons
- Directory: Save to temporary file
- File prefix

At the bottom are buttons for "Generate query", "Run query", "Overpass Turbo", and a "Documentation" dropdown menu.

³ https://wiki.openstreetmap.org/wiki/Overpass_API

In the layers window, right click on the new layers, open “Layer Properties” and select the tab “Labels”, choose “Single Labels” and select a name attribute. Do this for all layers.



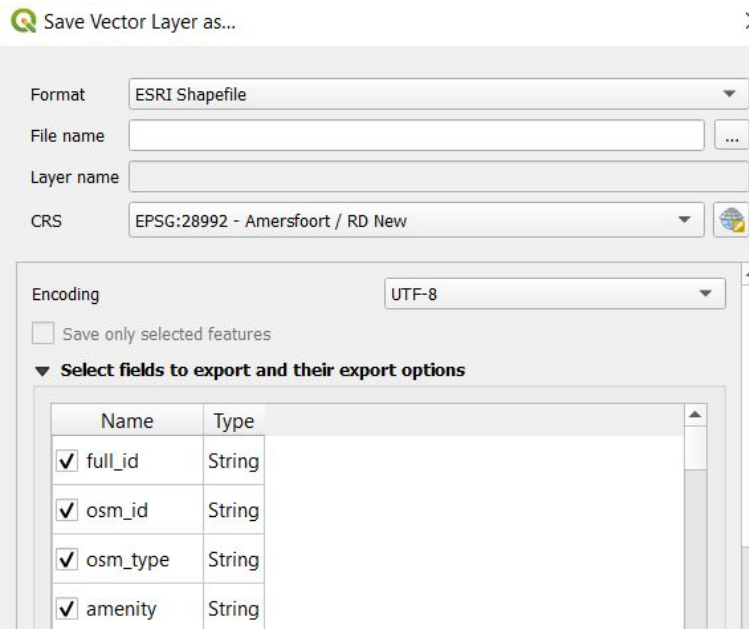
You should now zoom into Amsterdam and take a closer look at the schools there. As you can see there are some equivalent schools in OSM and schoolwijzer data. Also, the Open Topo background map likewise contains some information about schools (e.g. “Linneaus”):



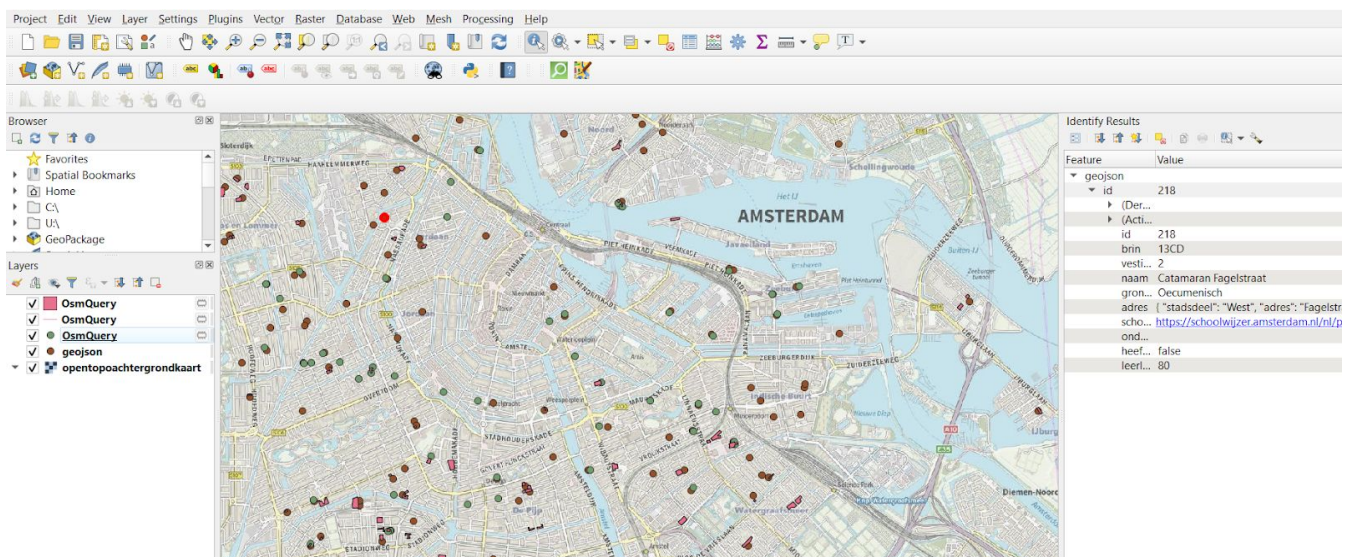
Comparison of 4 school layers: schoolwijzer (yellow), OSM points (light green), OSM polygons (dark green), topographic background map.


Save the datasets as shapefiles, and project them into “RD new”

The Vector layers that were loaded so far are still in the original format and in an “unprojected” CRS, which is WGS84. For any geometric computations, it is therefore essential to reproject them all into the same Dutch CRS “RD new” first. Also, we will all turn them into the common shapefile format. Click on each dataset, open the context menu and choose “Export/Save Feature As”.




Choose RD new and ESRI Shapefile and keep all attributes selected in the export. Then save each file locally and reload it into the map window. Remove the old layers. The result should look like this:

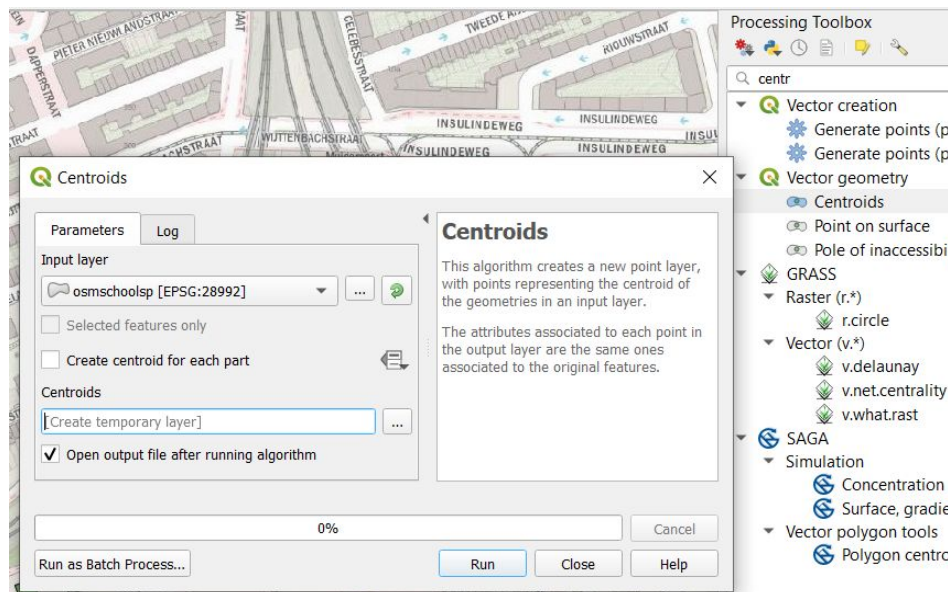


You can click on “Identify features” () in the menu to look up attribute values for a given entity displayed in the map window. Make sure you click on the respective layer first.

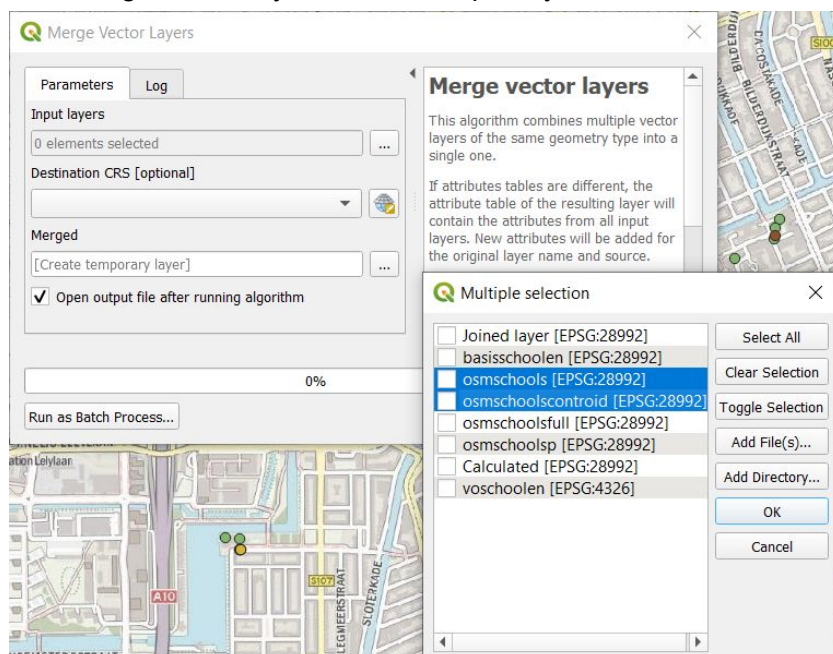
Make a single point layer out of the two OSM layers

Since the two OSM layers (point, polygon) each contain distinct schools, we need to turn them into a single layer in order to assess completeness and other quality dimensions. This can be done by first turning the polygon layer into a point layer of **centroids**.

In QGIS, open the Processing toolbox () and search for the “Vector geometry” tool “Centroids”. Choose the OSM polygon layer as input and choose an output layer name under “Centroids”:




Then merge the OSM centroid layer and the OSM point layer into a single layer, using the tool “Merge vector layers”. Press “Input layers”, and select the right OSM layers:



Call the new layer e.g. “osmschoolsfull” and add it to the map window.

Determine the extent of a layer

To determine the extent of a layer, we make use of Python script. You can run Python

scripts directly within QGIS. Open the Python console () from the menu and paste the following code, which accesses the currently active layer using the **iface** object:

```
layer = iface.activeLayer() # load the currently active layer
ext = layer.extent()
xmin = ext.xMinimum()
xmax = ext.xMaximum()
ymin = ext.yMinimum()
ymax = ext.yMaximum()
coords = "%f,%f,%f,%f" %(xmin, xmax, ymin, ymax) # this is a
string that stores the coordinates
print(coords)
```

```
24>>> layer = iface.activeLayer() # load the layer as you want
25>>> ext = layer.extent()
26>>> xmin = ext.xMinimum()
27>>> xmax = ext.xMaximum()
28>>> ymin = ext.yMinimum()
29>>> ymax = ext.yMaximum()
30>>> coords = "%f,%f,%f,%f" %(xmin, xmax, ymin, ymax) # this is a string that stores the coordinates
31>>> print(coords)
32 113184.014794,130739.162146,479733.133163,491602.574842
```

Now let's prepare an **external Python script** for assessing the geodata quality. Open PyCharm with the QGIS configuration, as described earlier (*PyQGIS.bat*). Then generate a new Python script "quality.py", and enter the following imports:

```
import os
from qgis.core import *
from qgis.core import *
from qgis.gui import *
from qgis.utils import *
from qgis.PyQt.QtCore import *
from qgis.PyQt.QtGui import *
from qgis.core import *
from math import sqrt
```

Then we need to set up the QGIS bridge at the beginning of the script. **Note:** Adjust the "local" path to your local data folder, and the "prefix" path to your local QGIS installation:

```
# Create a reference to the QgsApplication. Setting the
# second argument to False disables the GUI.
qgs = QgsApplication([], True)
# Load providers
```



```

qgs.initQgis()
local=r"C:\Users\schei008\surfdrive\Shared\ADScourse"
# [SC] Supply path to qgis location
QgsApplication.setPrefixPath("C:/Program Files/QGIS
3.10/apps/qgis", True) #

```

At the end of your script, close the bridge again:

```

# Finally, exitQgis() is called to remove the
# provider and layer registries from memory
exitcode = qgs.exec_()
qgs.exitQgis()
sys.exit(exitcode)

```

Now write a small function that determines the extent for any layer:

```

def getExtent(layer):
    ext = layer.extent()
    xmin = ext.xMinimum()
    xmax = ext.xMaximum()
    ymin = ext.yMinimum()
    ymax = ext.yMaximum()
    coords = "%f,%f,%f,%f" % (xmin, xmax, ymin, ymax) # this
    is a string that stores the coordinates
    print("Extent is "+coords)
    return coords

```

Then load all your local vector layers, using the following code, adapted to your respective data files:

```

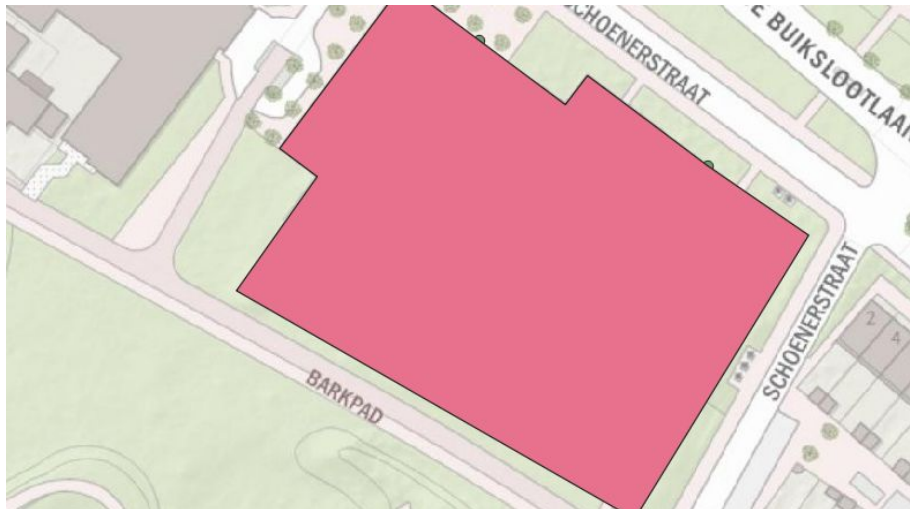
vlayer = QgsVectorLayer(os.path.join(local, "basisschoolen.shp"),
"Basisschoolen", "ogr")

```

Finally, test your script by applying the `getExtent` function to these layers.



Determine the resolution of a (polygon vector) layer

Think a moment how you could assess the resolution of the OSM polygon dataset. To assess the resolution, you need to measure the size of the smallest distinguishable length of the segments that constitute the geometries of a layer. In the OSM polygon layer, these segments are used to depict the corners of buildings:



This means we need a script that searches through all segments. Remember that the OSM geometries are coded as **MultiPolygons**.

Here are examples of Multipolygons in the WKT encoding standard⁴:

MultiPolygon		MULTIPOLYGON (((30 20, 45 40, 10 40, 30 20)), ((15 5, 40 10, 10 20, 5 10, 15 5)))
		MULTIPOLYGON (((40 40, 20 45, 45 30, 40 40)), ((20 35, 10 30, 10 10, 30 5, 45 20, 20 35), (30 20, 20 15, 20 25, 30 20)))

As you can see, Multipolygons are encoded as a nested array. The first array of a multipolygon consists of a list of **polygons with or without holes**. Each of the latter, in turn, consists of an array of at least one boundary ring as a first element, and potentially further rings that represent holes of this polygon. On the lowest level, each **ring** is a “closed” array of coordinate pairs, such that the last coordinate is equal to the first one. Thus we need to search through a nested array of depth 3 to find all segments.

⁴ https://en.wikipedia.org/wiki/Well-known_text_representation_of_geometry

So, first, add a Python function to your script that can be used to search the smallest length inside a ring given as array:

```
def getsmallestlength(ring, smallestlength):
    lastpoint = None
    for point in ring:
        length = smallestlength
        if lastpoint is not None:
            length = sqrt(point.sqrDist(lastpoint))
        lastpoint = point
        if length < smallestlength:
            smallestlength = length
    return smallestlength
```

This function loops through the given input ring (which is an array), measures the Euclidean distance between consecutive point pairs and returns the smallest segment length in the ring. Then define a function `getResolution` to assess the resolution over all rings in a given layer:

```
def getResolution(layer):
    features = layer.getFeatures()
    smallestLength= 1000
    nofeatures= 0
    for feature in features:
        geom = feature.geometry()
        nofeatures +=1
        geotype = ""
        if geom.wkbType() == QgsWkbTypes.MultiPolygon:
            geotype = "MultiPolygon"
            x = geom.asMultiPolygon()
            numPts = 0
            for polyg in x:
                for ring in polyg:
                    numPts += len(ring)
                    smallestLength =
                        getsmallestlength(ring,smallestLength)

    print("smallest distinguishable length : " +
        str(smallestLength) + " in " + str(nofeatures) + " " +
        geotype+" features")
```

Before accessing the properties of a geometry, we need to first cast it to the right type using `asMultiPolygon()`. To do this safely, we can first test for the geometry type using the `wkbType()` function. Implement this function and determine resolution of OSM polygon data.

Measuring completeness and spatial accuracy

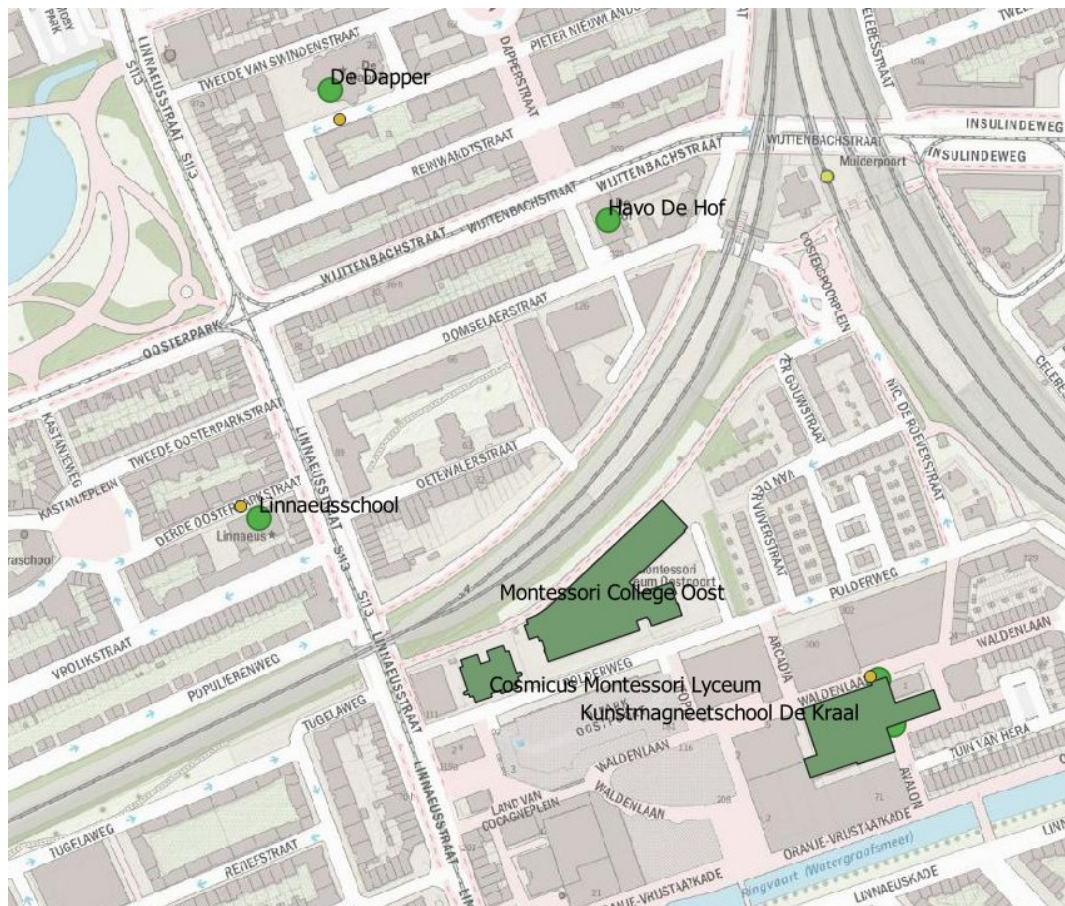
In order to measure completeness and spatial accuracy, we need a **reference data set**. For our purpose, we will compare data against the schoolwijzer data as reference to assess the quality of the OSM data, because it is an authoritative source. We start with a qualitative, manual inspection of the quality. Then we proceed to a quantitative analysis by matching these two datasets.

Qualitative analysis

To get a first impression of the data quality, we can compare the data sources by visual inspection. For this purpose, zoom to a sufficient level of detail so that you can see the school labels also in the underlying WMTS layer for Amsterdam. Then visually match the labels of OSM data, schoolwijzerdata with the information given in the underlying WMTS layer.

What is your impression of the completeness of the OSM data source with respect to the other two layers?

What is your impression of the spatial accuracy of both the schoolwijzerdata and the OSM data, as compared to the (more detailed) topographic WMTS layer?



Matching schools based on distance and similarity of names

In order to measure these quality dimensions, we have to first **match** data items in the reference dataset with **equivalent** items in the OSM data set. Thus the difficulty lies in assessing, whether two schools are considered equivalent or not.

The screenshot shows the 'Join Attributes by Nearest' dialog box in QGIS. The 'Parameters' tab is active. The 'Input layer' is set to 'basisscholen [EPSG:28992]' and 'Input layer 2' is set to 'osmschoolsfull [EPSG:28992]'. The 'Layer 2 fields to copy' section is empty. The 'Maximum nearest neighbors' is set to 1, and the 'Maximum distance' is set to 200,000,000 meters. The 'Joined layer' is set to '[Create temporary layer]'. The 'Open output file after running algorithm' checkbox is checked. The 'Unjoinable features from first layer' is set to '[Skip output]'. The 'Open output file after running algorithm' checkbox is unchecked. The 'Run as Batch Process...' button is visible at the bottom left. The 'Run', 'Close', and 'Help' buttons are at the bottom right. A progress bar at the bottom shows 0% completion. A help panel on the right titled 'Join attributes by nearest' provides a detailed description of the algorithm.

Join attributes by nearest

This algorithm takes an input vector layer and creates a new vector layer that is an extended version of the input one, with additional attributes in its attribute table.

The additional attributes and their values are taken from a second vector layer, where features are joined by finding the closest features from each layer. By default only the single nearest feature is joined, but optionally the join can use the n-nearest neighboring features instead. If multiple features are found with identical distances these will all be returned (even if the total number of features exceeds the specified maximum feature count).

If a maximum distance is specified, then only features which are closer than this distance will be matched.


The output features will contain the selected attributes from the nearest feature, along with new attributes for the distance to the near feature, the index of the feature, and the coordinates of the closest point on the input feature (feature_x, feature_y) to the matched nearest feature, and the coordinates of the closest point on the matched feature (nearest_x, nearest_y).

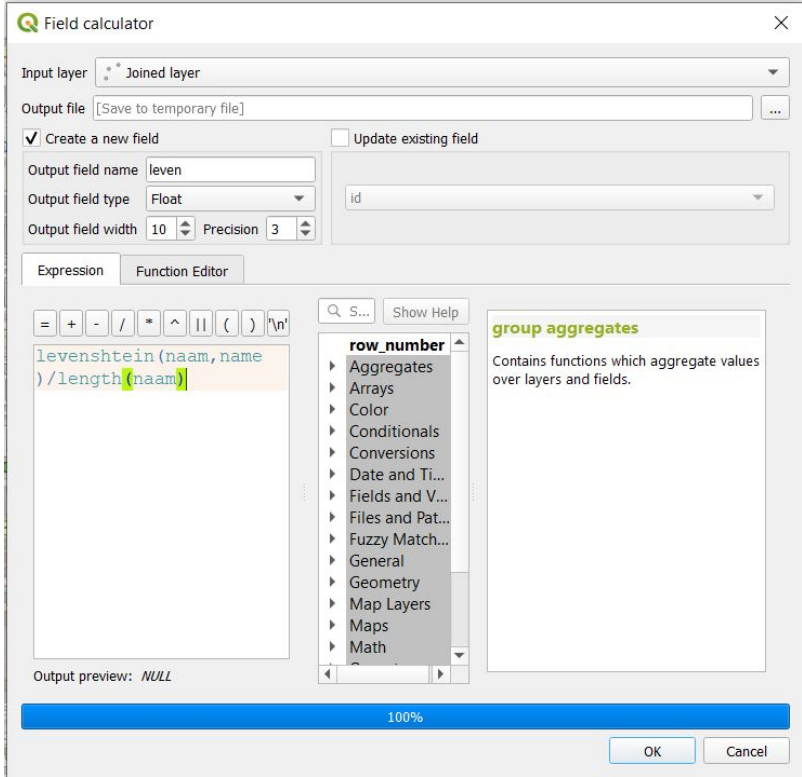
This algorithm uses purely Cartesian calculations for distance, and does not consider geodetic or ellipsoid properties when determining feature proximity.

To define equivalence, we make use of spatial distance and similarity of school names. Open the tool “Join Attributes by nearest”. This tool can be used to join entities in a layer by the nearest entities in another layer. Select the schoolwijzer layer (*basisscholen*) as the first layer and the full OSM layer as second layer in the join method. Set “maximum nearest neighbors” to 1 and assume a “maximal join distance” of 200 meters. As a result, the new joined layer will be added as a virtual layer to the map. Open the attribute table of this new joined layer, via the context menu:

Joined layer :: Features Total: 277, Filtered: 277, Selected: 0											
	full_id	osm_id	osm_type	amenity	name	vestigings	naam	grondslag	adres	distance	feature_x
209	w296847984	296847984	way	school	3e Dalton	0	3e Daltonschoo...	Public	{ "stadsdeel": "Z...	29,5088244943...	121.646,09057
210	w296847981	296847981	way	school	Lycée Vincent v...	0	3e Daltonschoo...	Public	{ "stadsdeel": "Z...	61,6181172475...	121.649,73448
211	w296847980	296847980	way	school	NULL	0	3e Daltonschoo...	Public	{ "stadsdeel": "Z...	47,9731206475...	121.709,01208
212	w373953911	373953911	way	school	NULL	NULL	NULL	NULL	NULL	NULL	NA
213	w373953853	373953853	way	school	Damstede Lyce...	NULL	NULL	NULL	NULL	NULL	NA
214	w323875518	323875518	way	school	Rosj Pina	0	Rosj Pina	Jewish	{ "stadsdeel": "Z...	56,0779762829...	120.904,63488
215	w323875517	323875517	way	school	Maimonides	0	Rosj Pina	Jewish	{ "stadsdeel": "Z...	62,6673214366...	120.907,01558
216	w277263063	277263063	way	school	Berlage Lyceum	NULL	NULL	NULL	NULL	NULL	NA
217	w277153337	277153337	way	school	NULL	0	2e Daltonschoo...	Public	{ "stadsdeel": "Z...	24,3116541348...	119.784,66702
218	w277153327	277153327	way	school	NULL	0	Schoolverenigin...	General Special	{ "stadsdeel": "Z...	27,3001606716...	119.786,20754
219	w277123466	277123466	way	school	Fons Vitae Lyce...	0	Peetersschool	Catholic	{ "stadsdeel": "Z...	35,6856976797...	120.326,39695

The attribute “name” originates from OSM, while the attribute “naam” originates from the schoolwijzer data. The join also adds a new column “distance”. We use these three attributes now to increase the quality of our matching.

First, we compute a new column with the **Levenshtein distance** between the two school names. For this purpose, open the Field Calculator (), and enter the following expression to compute the values of a new attribute:



Field calculator

Input layer: Joined layer

Output file: [Save to temporary file]

☒ Create a new field ☐ Update existing field

Output field name: leven

Output field type: Float

Output field width: 10 Precision: 3

Expression: levenshtein(naam, name) / length(naam)

Output preview: NULL

100%

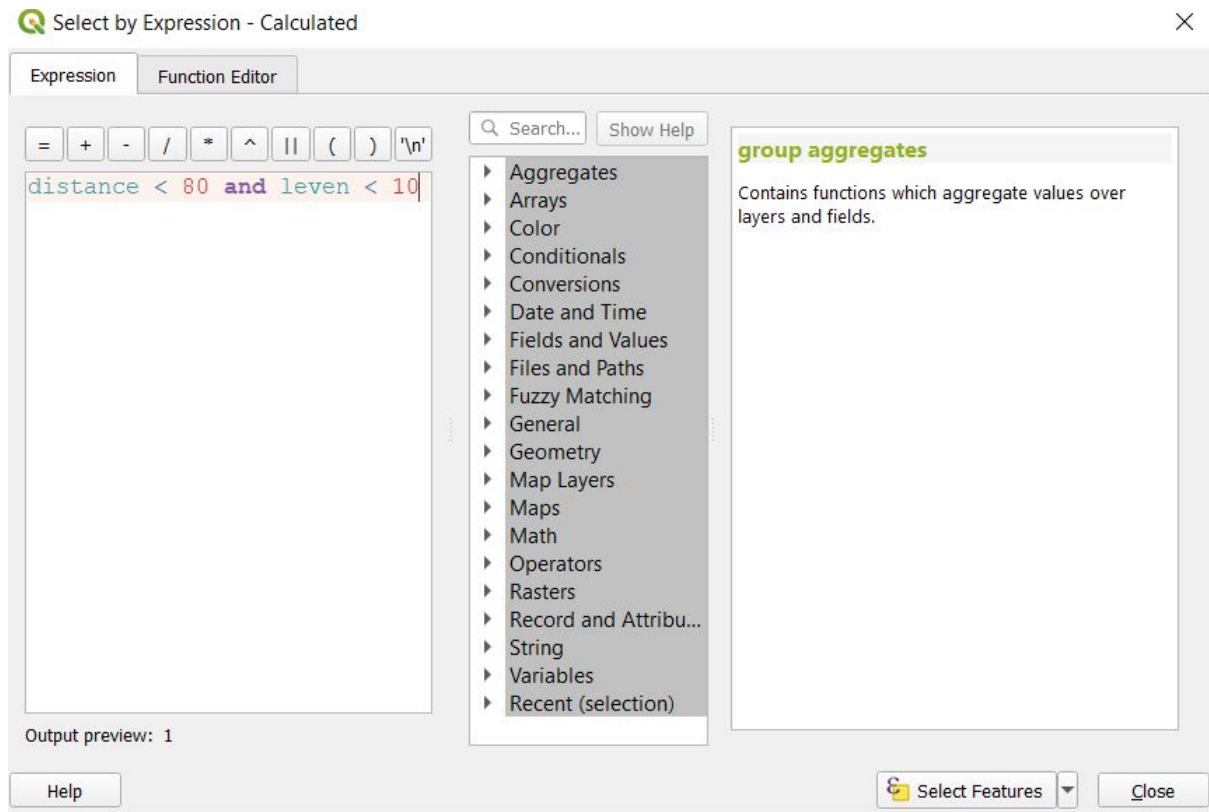
OK Cancel

This expression computes the Levenshtein distance (corresponding to the number of edits necessary to transform a given string into another one, see⁵), and standardizes it by the length of one of the two strings. You may improve this formula by dividing by the longest of the two strings.

⁵ https://en.wikipedia.org/wiki/Levenshtein_distance

Once the new attribute is computed, it is stored in a new layer (Calculated). We finally filter out invalid matches by a maximal spatial and levenshtein distance as filter condition, using

“Select by expression” ():

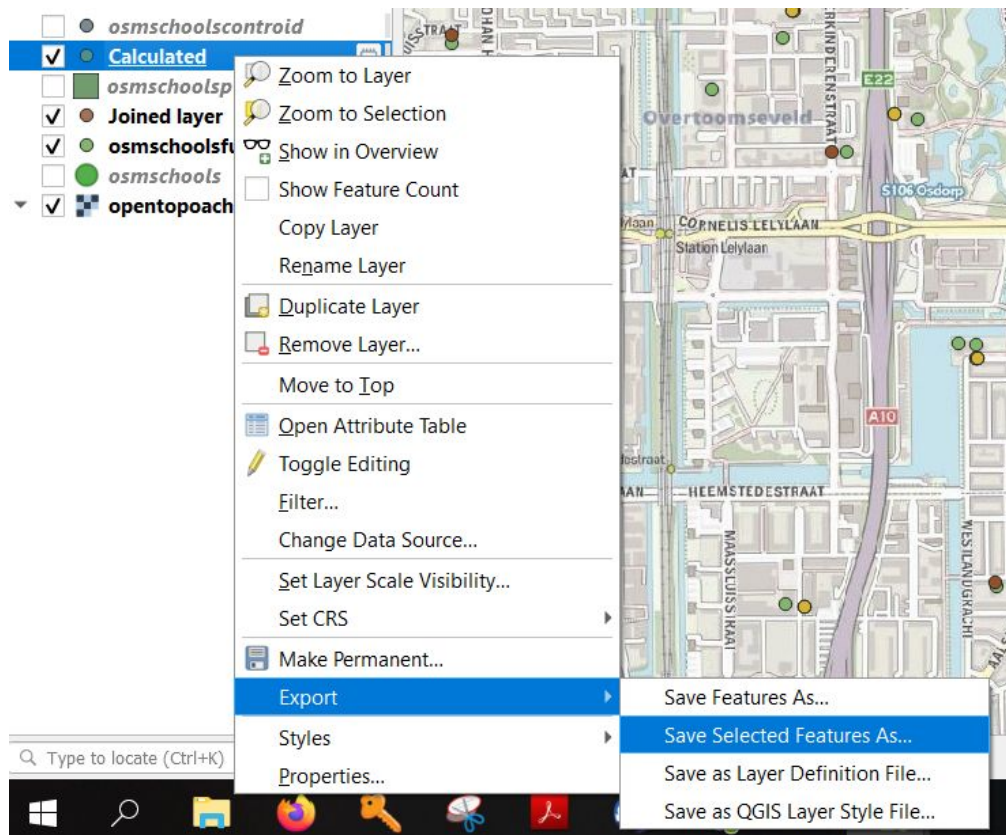


Try out several filter expressions and check whether the resulting table really contains valid matches or not. Note that distance is in meters, while “leven” is below 1.

Calculated :: Features Total: 234, Filtered: 234, Selected: 234

	id	brin	vestigings	naam	grondslag	adres	schoolwijz	onderwijs	heeft_voor	leerlingen
1	147	20XT	0	De Spaarndammerhout	Public	{ "stadsdeel": "...	https://schoolwi...	Dalton	1	268
2	156	20ZE	0	De Vier Windstreken	Public	{ "stadsdeel": "...	https://schoolwi...	NULL	1	309
3	117	20SL	0	De Toekomst/ Spring High	Public	{ "stadsdeel": "...	https://schoolwi...	NULL	1	250
4	150	20XZ	0	De Tamboerijn	Public	{ "stadsdeel": "Z...	https://schoolwi...	NULL	1	434
5	128	20TX	3	De Stern, 12e Montessorischool	Public	{ "stadsdeel": "Z...	https://schoolwi...	Montessori	1	49
6	126	20TX	1	9e Montessorischool Scholekster	Public	{ "stadsdeel": "Z...	https://schoolwi...	Montessori	1	258
7	170	22LE	0	8e Montessorischool Zeeburg	Public	{ "stadsdeel": "...	https://schoolwi...	Montessori	1	301
8	129	20UP	0	7e Montessorischool	Public	{ "stadsdeel": "...	https://schoolwi...	Montessori	1	494
9	125	20TX	0	6e Montessorischool Anne Frank	Public	{ "stadsdeel": "Z...	https://schoolwi...	Montessori	1	347
10	175	23HR	0	Al Maes	Islamic	{ "stadsdeel": "...	https://schoolwi...	NULL	1	499
11	177	30UF	0	Al Jawhara	Islamic	{ "stadsdeel": "...	https://schoolwi...	NULL	1	323
12	14	07JH	0	Admiraal de Ruyter	Protestant Chris...	{ "stadsdeel": "...	https://schoolwi...	NULL	1	228
13	41	12WS	0	Achtersprong	Philosophical	{ "stadsdeel": "Z...	https://schoolwi...	NULL	1	273

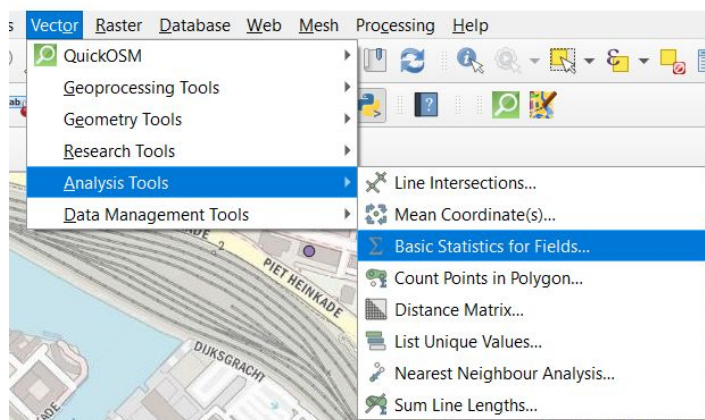
Save the matched features by clicking on the filtered layer “Calculated”, and then select Export/Save Selected Features As ...



Assess completeness and spatial accuracy

Compare the location data also with Open Topo. You will see that the OSM data is spatially more accurate but rather incomplete with respect to schoolwijzer data. Then assess the completeness and spatial accuracy of the OSM datasets with respect to schoolwijzerdata:

- Completeness (percentage): Count the number of matched schools and divide by total number of schools in schoolwijzer
- Spatial accuracy : Compute the average distance between matches. To compute the average, use the tool “Basic Statistics for Fields ...” accessible over the “Vector/Analysis Tools” Menu.



Assignment: assess quality of one other geodata source, and document everything

Do a similar kind of analysis for *at least one other geodata source*. This can be a polygon dataset, so that you can reuse your script (otherwise you would need to adapt your script), or it could be a point or line data source. You might also use a different OSM key/value pair to obtain OSM vector data, or reuse a geodata source you have collected in lab 1.1.

If you don't find any good reference dataset as above (and in order to save time), you can do a *qualitative assessment* of spatial accuracy and completeness, by just visually comparing it with some background map.

Finally **document all your quality assessments** for all datasets in a pdf document, including references to the data sources, some screenshots of maps and some short text describing each data quality dimension for each source. Submit this document on blackboard.