Utrecht University

Applied Data Science Master's degree programme

Spatial Data Analysis and Simulation Modelling course

Instruction manual for Lab 1.1: **Retrieval and CRS**

Document version: 1

Document modified: 2020.10.14

Dr. Enkhbold Nyamsuren, Simon Scheider

Department of Human Geography and Spatial Planning

Faculty of Geosciences

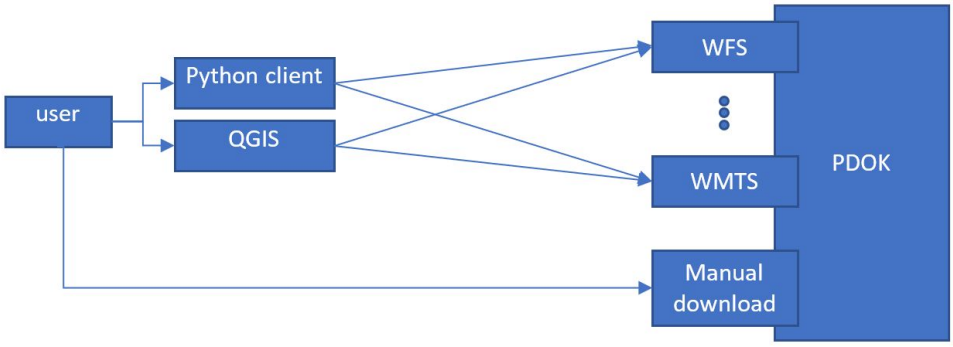
s.scheider@uu.nl

Table of Contents

Retrieving PDOK data with Web Feature Service

There are two common ways to retrieve dataset from repositories: manually and programmatically. While manual retrieval is easy and intuitive, it is not as efficient as programmatical retrieval. With programmatical retrieval, dataset can be queried for, filtered, and retrieved automatically on demand. Furthermore, GIS and other data analysis software can have built-in tools for programmatically retrieving dataset from common data repositories. While not all web repositories support programmatical retrieval, it is becoming a norm to do so. To support programmatical retrieval, a web repository should expose a software interface that can be accessed by users. A user should have a tool, aka a client, that can connect to this interface to send various request, such as a request to download a dataset.

Interfaces are commonly implemented as APIs, and web services are specific form of web APIs. Note that web services may serve different purposes that are not limited to dataset retrieval. For example, PDOK has about seven different web services (the full list is here https://www.pdok.nl/services-en-api-s) such as a web service for downloading data and a web service for retrieving map images. Depending on its purpose, a web service should follow a specific standard. Such standard ensures that different clients know how to connect to and use the web service. For example, Web Map Tile Service (WMTS) is a standard for a web service for retrieving map images. Web Feature Service (WFS) is a standard for web services for downloading geographical vector data.



PDOK has a large collection of geographical datasets (https://www.pdok.nl/datasets) and supports three different methods of data retrieval: RESTful API, Geo Services, and Downloads. For example, in the figure below, the PDOK dataset *Basisregistratie Topografie (BRT) TOPNL* supports all three methods of retrieval. However, note that not all methods of retrieval may be available for each dataset. Downloads is for manual retrieval, and RESTful API and Geo Services are for retrieval via web services. Geo Services is for WFS-based retrieval, and it is the most commonly available method among PDOK datasets. We will further explore how to retrieve PDOK data via its WFS with Python client and with QGIS's built-in client tool.

1. Basics of WFS

To connect any client to WFS, we need to know, at first, few basic things on how WFS works. In PDOK, any dataset has a unique WFS URL. A client can use this WFS URL to connect to PDOK's WFS server and retrieve the dataset identified by the WFS URL. Note that WFS URL is not the same as the URL used to access dataset's webpage in a browser.

For example, the dataset Bestuurlijke Grenzen contains administrative boundaries of the Netherlands. The webpage of the dataset is https://www.pdok.nl/geo-services/-/article/bestuurlijke-grenzen. If we go to this webpage, we can find the WFS URL of the dataset:
1. Go to the dataset's webpage
2. Go to the Geo Services tab
3. WFS URL can be found in the table titled Bestuurlijke grenzen (WFS), in the row titled URL.

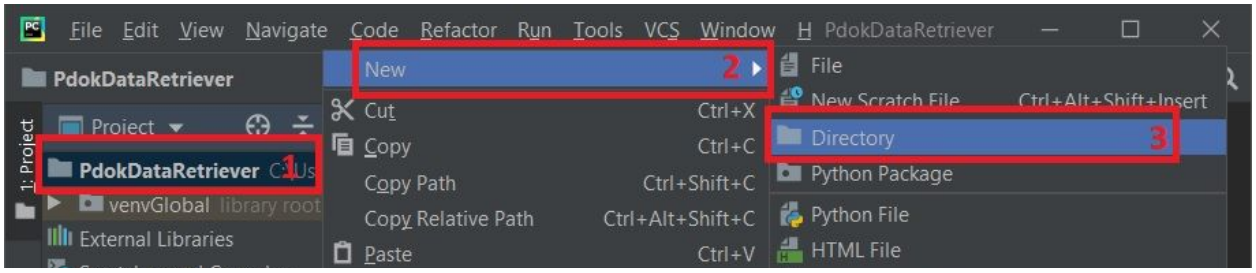
The row should contain a URL that looks like this:
https://geodata.nationaalgeoregister.nl/bestuurlijkegrenzen/wfs?request=GetCapabilities&service=wfs
Note that this is an HTTP query to the WFS server. The actual WFS URL is https://geodata.nationaalgeoregister.nl/bestuurlijkegrenzen/wfs. Everything after the question mark

"?" is a query string. The query string contains two parameters *request* and *service* and corresponding values for the parameters.

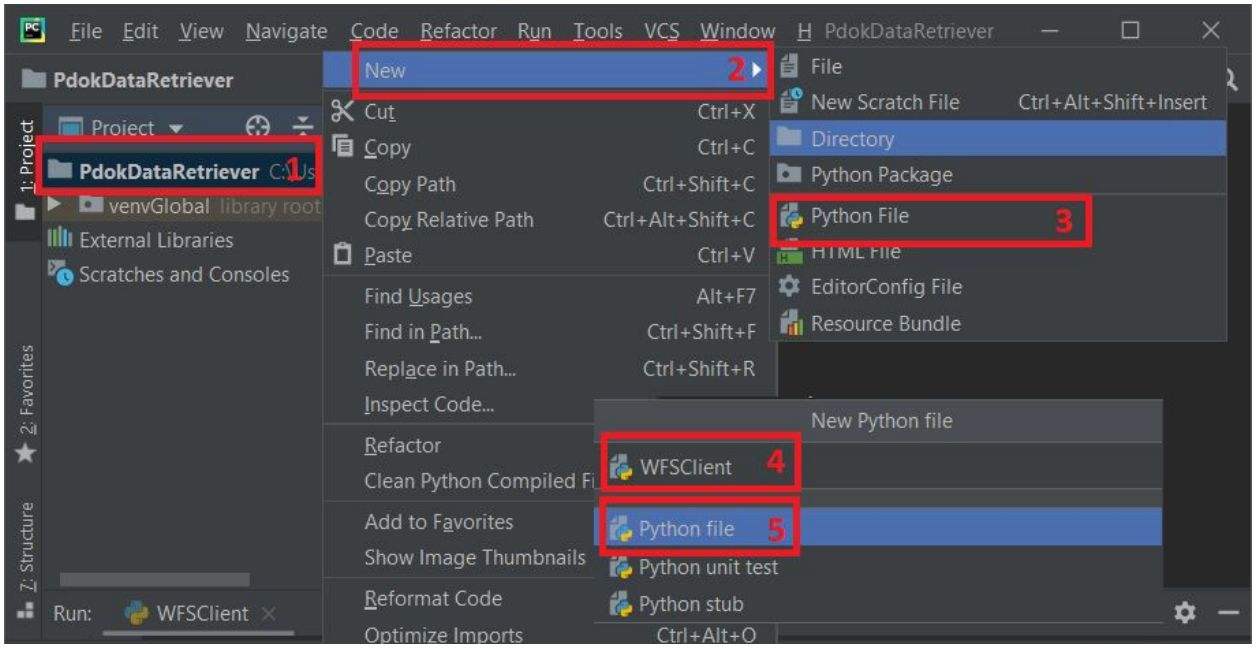
Apart from WFS URL, we also need to know the version of the WFS standard used in the WFS server. Different versions of the WFS standard have different query parameters. Based on the version, WFS client knows what kind of parameters it can use. PDOK servers supports versions up to WFS *2.0.0* as described in https://www.pdok.nl/wfs. However, other repositories may use different versions of the WFS standard.

2. Retrieving datasets with a Python client

First, we will create a new folder within the *PdokDataRetriever* project. We will store the retrieved datasets in this folder. Make sure you have the *PdokDataRetriever* project open in PyCharm.
1. In the project menu (left side), call the popup menu by a right mouse click on the root folder of the project.
2. In the popup menu, select the submenu *New*.
3. In the submenu, select the *Directory* option.
4. Name the new folder "*retrievedData*" and click *Ok*.



Next, we will create a new file to write down the source code for the WFS client.
1. In the project menu (left side), call the popup menu by a right mouse click on the root folder of the project.
2. In the popup menu, select the submenu *New*.
3. In the submenu, select the *Python File* option.
4. Name the new file *WFSClient*.
5. Double click on *Python file* option. The newly created file will automatically open. Now, we can write down some source code.
6. Copy/paste code from the file "WFSClient.py" (on blackboard) to the newly created file.



Let's discuss some of the key points in the code.

Import the `WebFeatureService` class from `owslib` library. An instance of this class becomes a client that connects to a WFS server.

```
from owslib.wfs import WebFeatureService
```

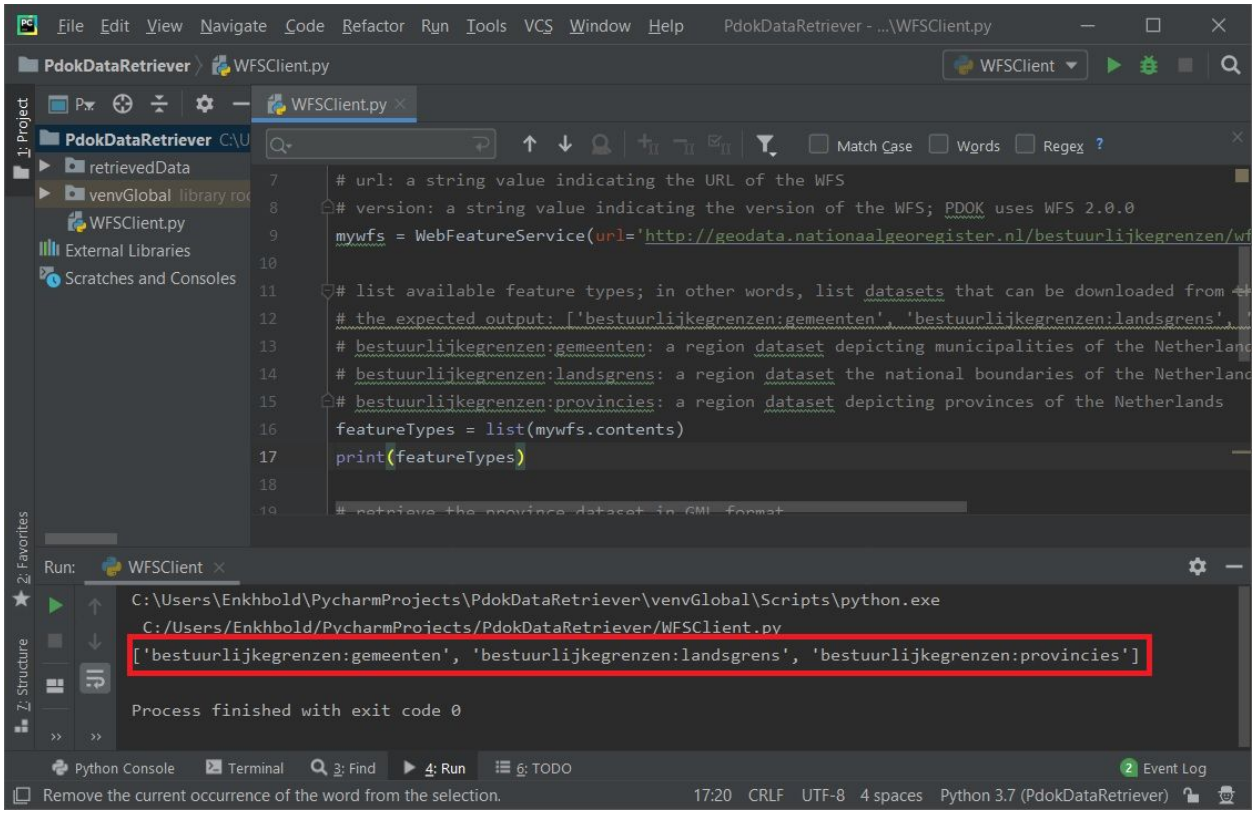
Instantiate the `WebFeatureService` class and store the instance in a variable named `mywfs`. Instantiation requires at least two parameters which are WFS URL and WFS version. Note that the version number should always include all three digits, and shortcuts such as "2.0" are not valid. `mywfs` is essentially our client. During the instantiation, it connects to the WFS server and retrieves basic information about the WFS server.

```
mywfs = WebFeatureService(
        url='http://geodata.nationaalgeoregister.nl/bestuurlijkegrenzen/wfs',
        version='1.1.0'
)
```

In this example, we are connecting to the WFS server that offers a dataset of Dutch administrative areas. It can be often the case that the dataset identified by the WFS URL is actually a collection of multiple datasets. For example, the dataset of administrative areas consists of three separate datasets: dataset of municipalities (*bestuurlijkegrenzen:gemeenten*), dataset of provinces (*bestuurlijkegrenzen:provincies*), and dataset of national boundaries (*bestuurlijkegrenzen:landsgrens*). In such cases, each dataset is referred to as a *feature type*. The code below retrieves the names of the available datasets (feature types) and prints them to the output panel.

```
featureTypes = list(mywfs.contents)
print(featureTypes)
```

Example of feature type names printed in the output panel:



Now, we can retrieve the dataset (feature type) by its name. For this, we can use the `getfeature` method. This method has one required parameter, which is `typename`. The value of this parameter should be the name of the dataset (feature type) you want to retrieve. `srsname` is an optional parameter and describes in which Coordinate Reference System (CRS) the dataset should be imported. All Dutch datasets should use the CRS named "*Amersfoort / RD New*" with the authority ID of "*EPSG:28992*". If this parameter is omitted then the dataset will be imported with its default CRS. `outputFormat` is an optional parameter. When the parameter is omitted, the PDOK WFS server by default returns the dataset in the Geography Markup Language (GML) format. If we want to have the dataset in a different format, for example JSON, then we can use the `outputFormat` parameter. The retrieved dataset is stored in the `boundaryFeatures` variable.

```
boundaryFeatures = mywfs.getfeature(
        typename = 'bestuurlijkegrenzen:landsgrens',
        srsname = 'urn:ogc:def:crs:EPSG::28992',
        outputFormat = 'json'
)
```

As a final step, the retrieved dataset is stored in a local file named "*boundaries.txt*". Note that this file is created inside the *retrievedData* folder that we have created previously.

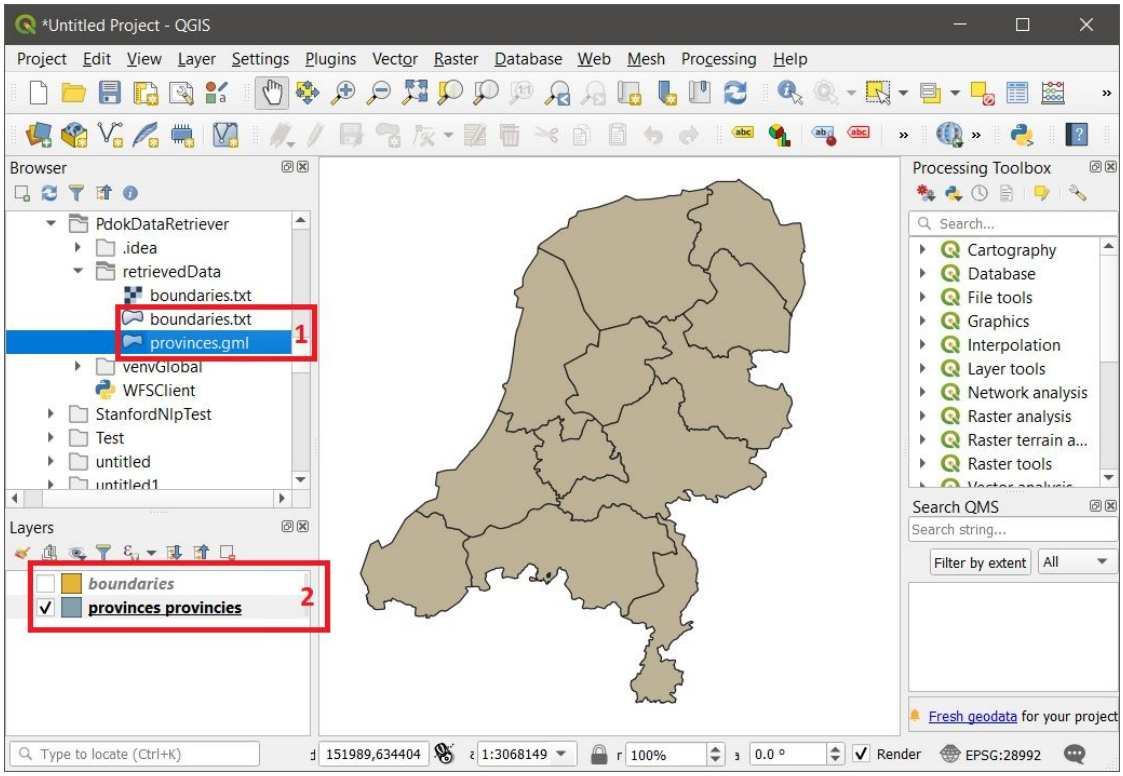
```
with open('retrievedData/boundaries.txt', 'wb') as boundaryOut:
    boundaryOut.write(boundaryFeatures.read())
```

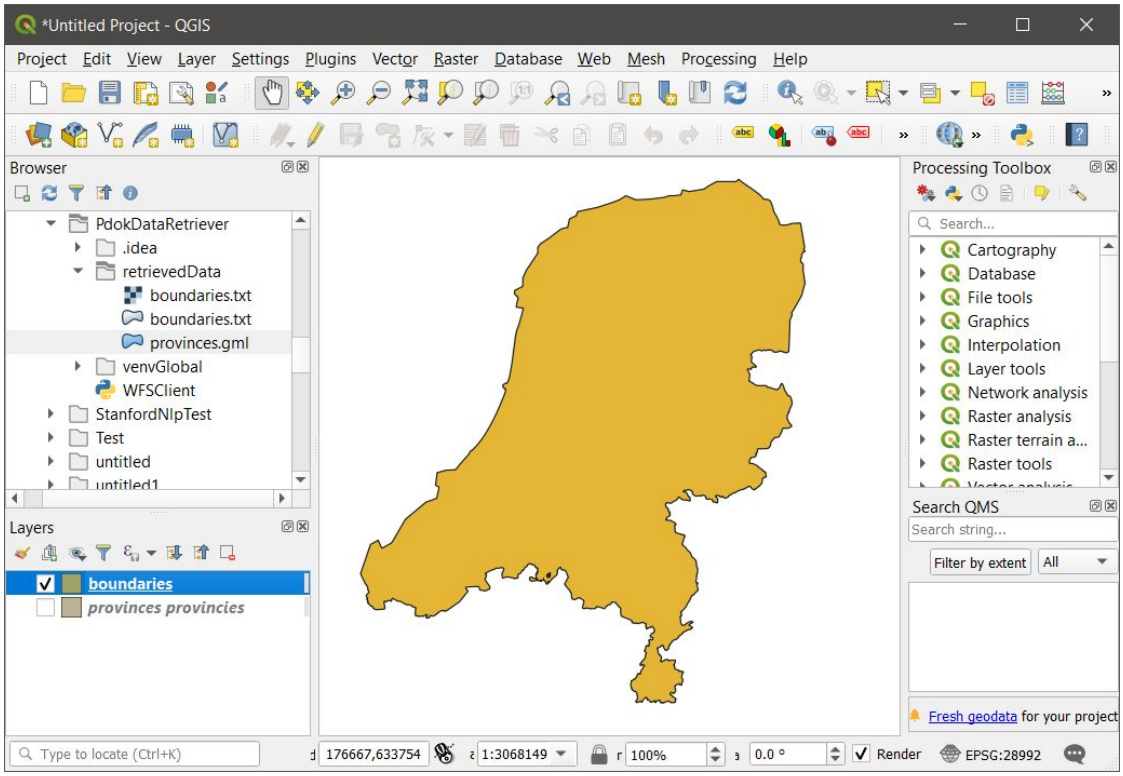
Now, we are ready to execute the Python client. To do so, go to "*Run -> Run 'WFSClient'*" in the main menu of PyCharm. The python code downloads two datasets and stores them in local files named

"*boundaries.txt*" and "*provinces.gml*" which are inside the *retrievedData* folder. The former is in JSON format, and the latter is in GML format.

3. Opening the retrieved datasets with QGIS

If the datasets were retrieved successfully and are properly formatted then we should be able to open them in QGIS. To start QGIS, go to "*QGIS 3.12 -> QGIS Desktop 3.12.2 with GRASS 7.8.2*" in the *Start* menu. Note that the version numbers may differ.
Once QGIS is open, use the *Browser* panel on the left side to navigate to the *retrievedData* folder where our Python client stored the dataset files. Double click on the dataset files to open them as map layers. Click Ok if you are prompted about coordinate transformation. Note that there might be two "*boundaries.txt*". Make sure to open the one with the icon of a polygon. The loaded layers will be listed in the *Layers* panel which is below the *Browser* panel.
The two figures below show example visualization of provinces and boundaries datasets. Note that colors may be different since color coding is chosen randomly and automatically.



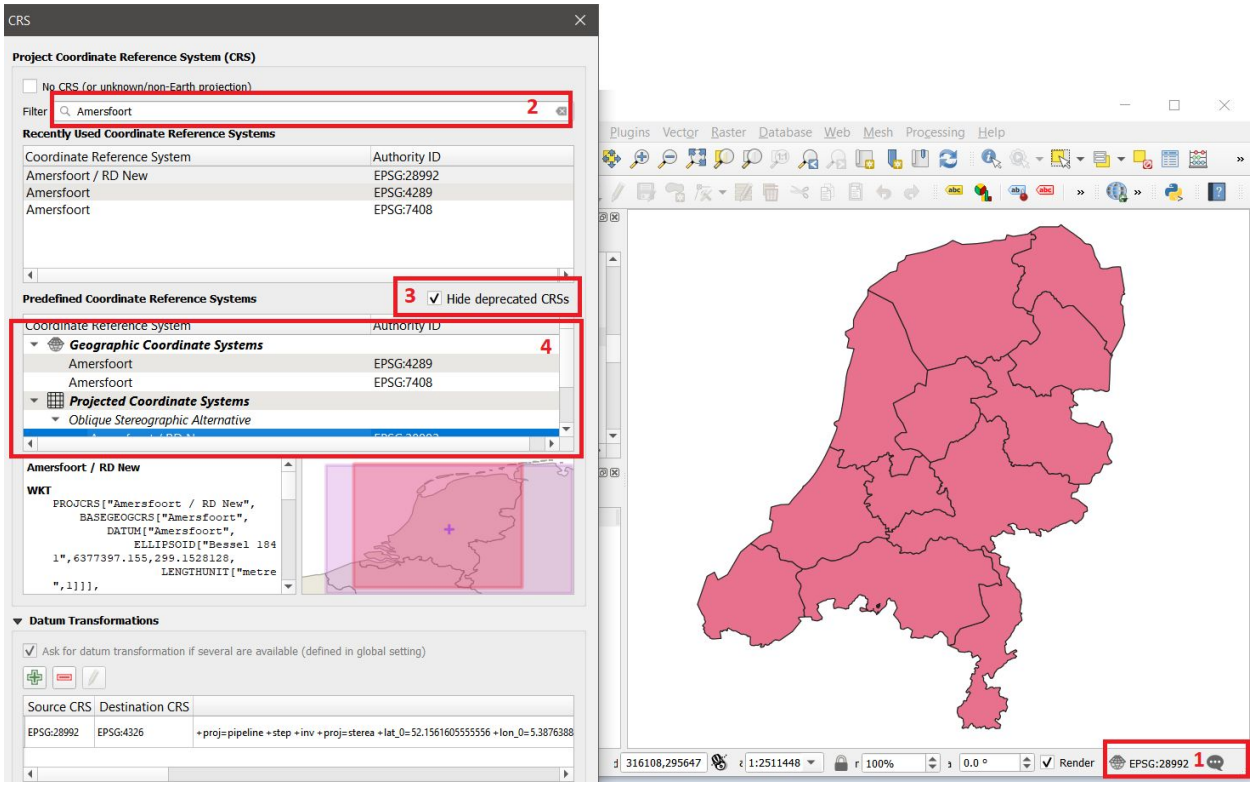


Working with Coordinate Reference Systems in QGIS

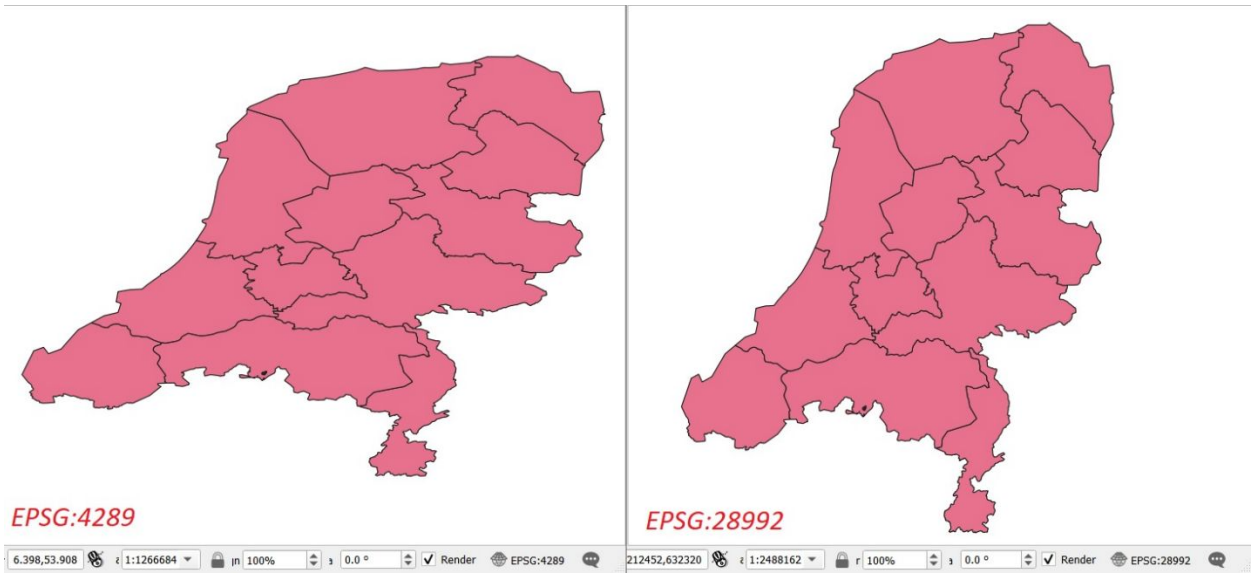
1. Setting the project and layer Coordinate Reference System (CRS) in QGIS

QGIS allows setting a Coordinate Reference System (CRS) that is best suitable for each individual project. For example, Dutch datasets often have the CRS named "*Amersfoort / RD New*" with the authority ID of "*EPSG:28992*". Different CRSs and their descriptions are listed on EPSG website: https://spatialreference.org/ref/epsg/.

To change the CRS of the project, go through the following steps:
1. The current CRS is visible at the bottom-right corner of the QGIS window. By default, the CRS should already be "*EPSG:28992*" for the datasets of Dutch national boundaries and municipalities. The CRS icon is also a clickable button. Click on it to open the CRS dialogue window.
2. In the *Filter* field, you can search for the desired CRS. For example, enter "*Amersfoort*".
3. Check "*Hide deprecated CRSs*" checkbox to remove irrelevant CRSs from the search results.
4. The relevant CRS search results are listed in the table "*Predefined Coordinate Reference Systems*". Note that the results include both geographic and projected CRSs. "*EPSG:28992*" is a projected CRS.
5. Try switching to a geographic CRS. Select "*Amersfoort EPSG:4289*" and press the *Apply* button. Changes in the CRS are reflected in changed shapes of the polygons reflecting the Dutch municipalities.



Below is the difference between two CRSs:



In QGIS, it is also possible to change CRS for an individual layer (dataset) or set the project CRS to the CRS of a layer:

1. Right click on the layer.
2. In the popup menu, go to "*Set CRS*".
3. "*Set Layer CRS...*" will open a CRS dialogue window where you can change the layer's CRS. "*Set Project CRS from Layer*" will change the project CRS to this layer's CRS.

2. Changing the dataset CRS and format with QGIS

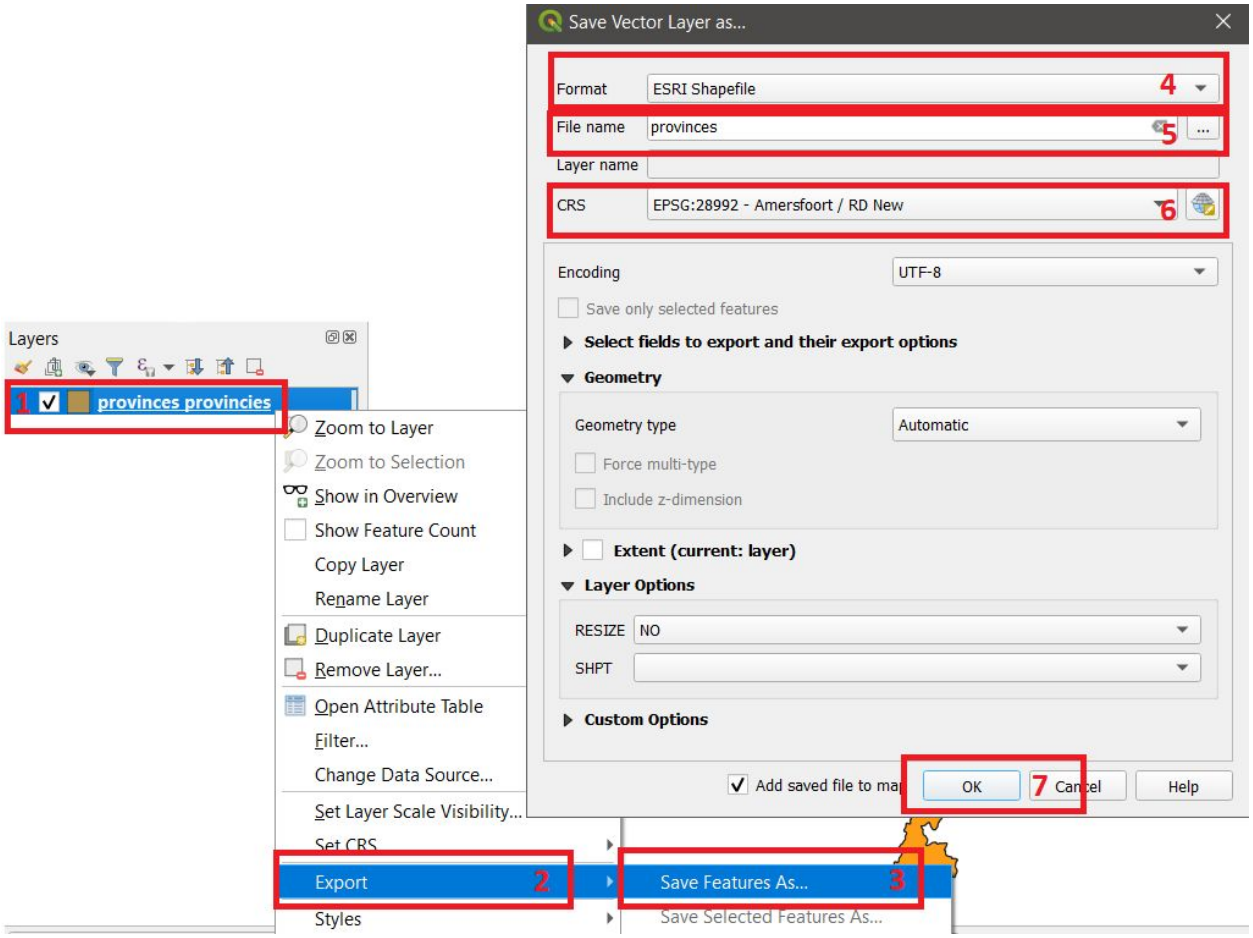
Having datasets with different CRS can cause various analytic problems. The safest option to avoid these problems is ensuring that all datasets have the same CRS. Note that we are referring to the CRS that is encoded into the dataset and not referring to the CRSs in QGIS layer and project.
For example, in the JSON dataset of Dutch national boundaries (*boundaries.txt*), the CRS is encoded as:

```
"crs":{
        "type":"name",
        "properties":{
                        "name":"urn:ogc:def:crs:EPSG::28992"
        }
}
```

The GML dataset of Dutch provinces (*provinces.gml*) has following xml tag encoding the CRS:

```
<gml:Polygon srsName="urn:ogc:def:crs:EPSG::28992" srsDimension="2">
```

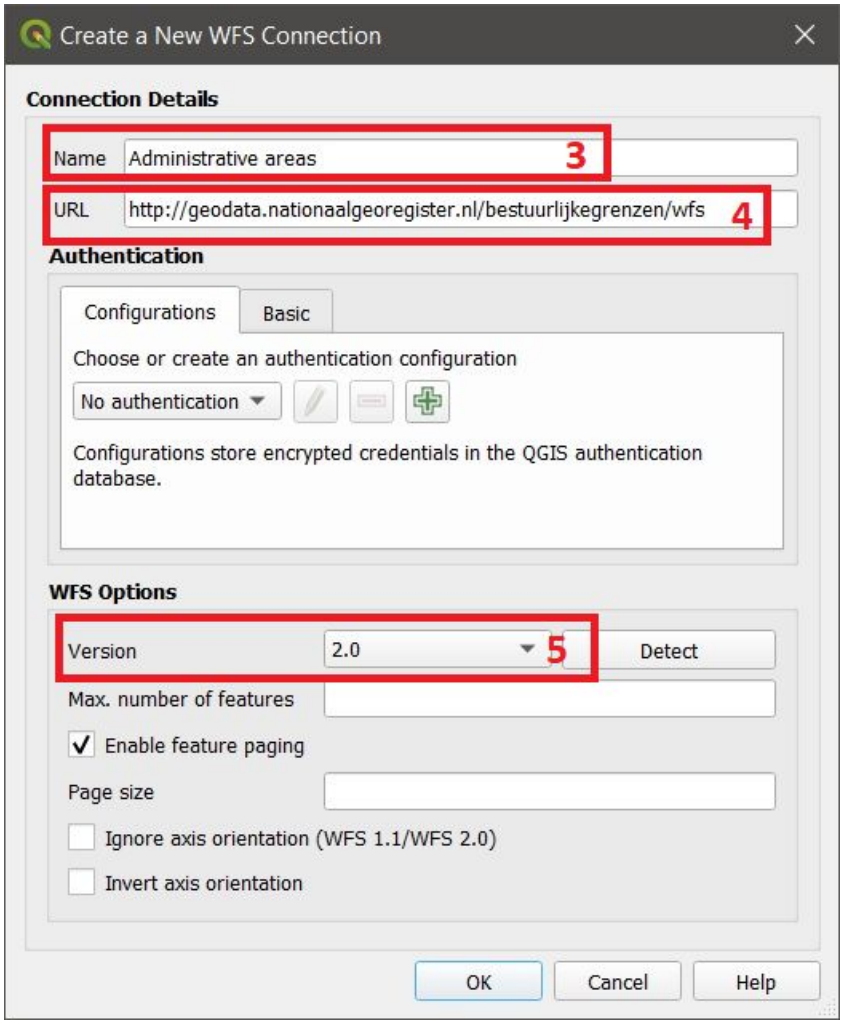
Let's assume you have the "*provinces.gml*" loaded into QGIS. To change its CRS and format follow the steps below:
1. Right click on the layer.
2. In the popup menu, go to the *Export* option.
3. Select "*Save Features As ...*" suboption to open the "*Save Vector Layer as ...*" dialogue window.
4. In the dialogue window, select the desired format. For example, "*ESRI Shapefile*" is the most popular format native to ArcGIS.
5. Enter the file name for the new dataset.
6. Select the correct CRS. All dataset of Dutch regions should use "*EPSG:28992 - Amersfoort / RD New*".
7. Click *OK* button to save the new dataset with new format and CRS.



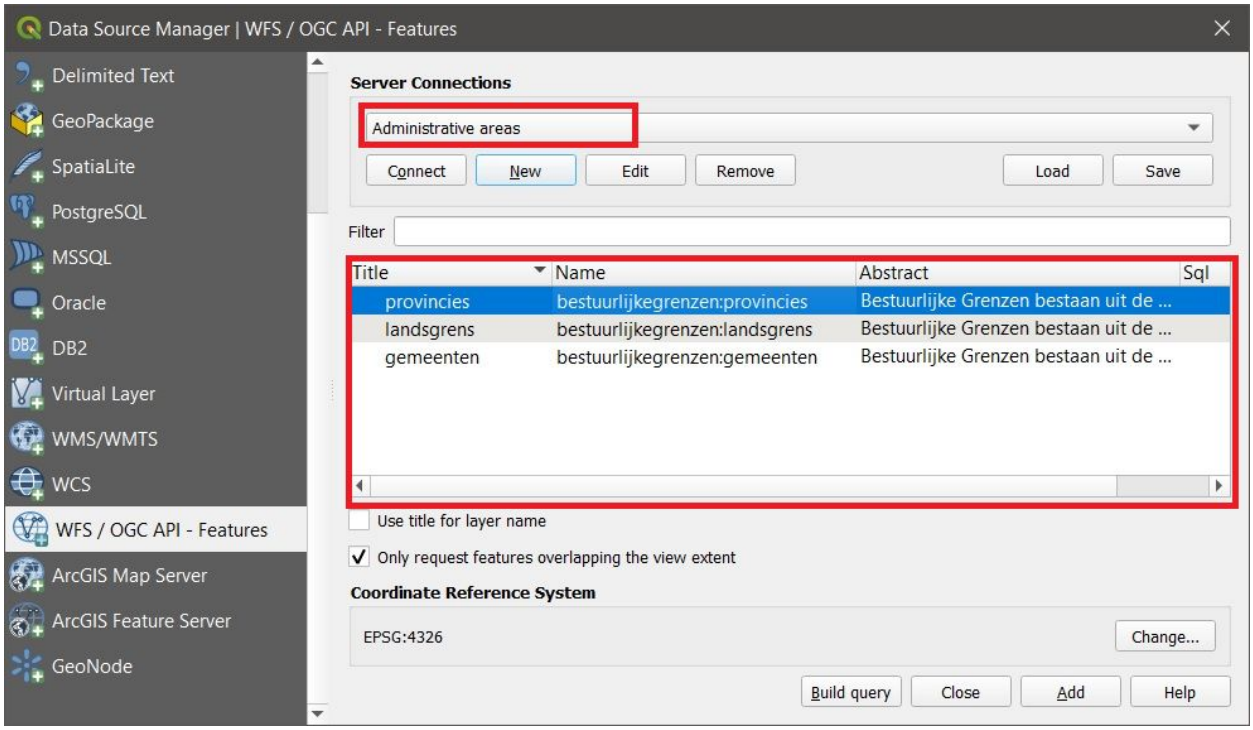
Retrieving datasets with QGIS's WFS tool on the example of PDOK

QGIS provides a convenient user interface for retrieving a dataset from a WFS server. In this guide, we will retrieve the same datasets of administrative areas but with QGIS.

1. In the main menu of QGIS, go to "*Layer -> Add Layer -> Add WFS Layer ...*". This will open the "*Data Source Manager*" window.
2. In the "*Data Source Manager*" window, click the *New* button. This will open another window named "*Create a New WFS Connection*".
3. Enter a name of your choice, for example, "*Administrative areas*".
4. Enter the WFS URL. In this case, it is http://geodata.nationaalgeoregister.nl/bestuurlijkegrenzen/wfs.
5. Pick the version number. For PDOK WFS, it should be "2.0".
6. Click Ok.



Now, you should see an update "*Data Source Manager*" window. "*Administrative areas*" should be listed under "*Server Connections*" section. Make sure it is selected, and click the *Connect* button. If the connection was successful, you will see the available dataset in the area below. These are the same three datasets that were listed by the Python client. Let's load all three datasets as layers in QGIS. Select each dataset and click the *Add* button at the bottom right corner. Close the "*Data Source Manager*" window. You should be able to see the loaded layers in the *Layers* panel of the main window.



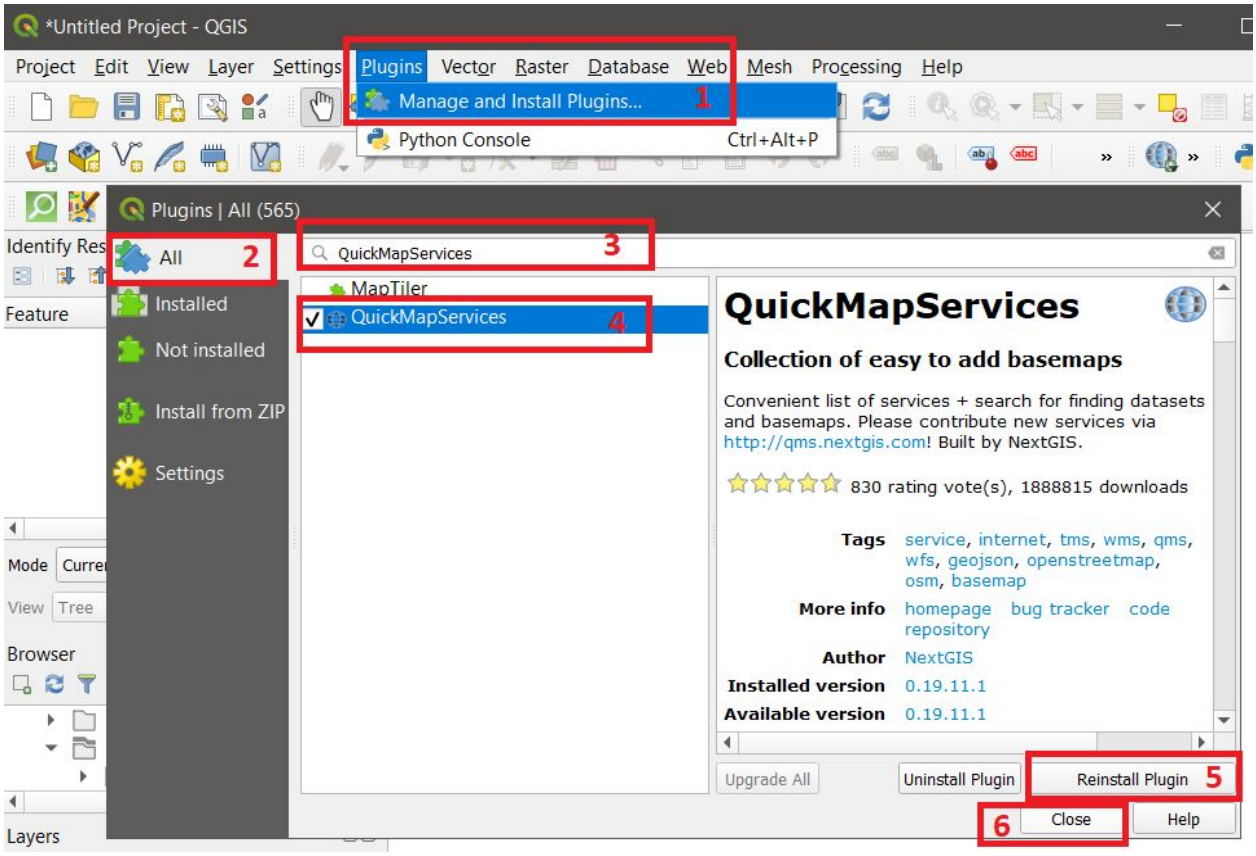
To store the layer to a local file, right mouse click on the layer to call the popup menu. In the popup menu, go to "*Export -> Save Features As...*". This will open the "Save Vector Layer as..." window. In the *Format* field, select the desired format, for example, "*Geography Markup Language [GLM]*". In the *File name* field, indicate the folder and the filename you want to save to. Click Ok.

Using map services in QGIS

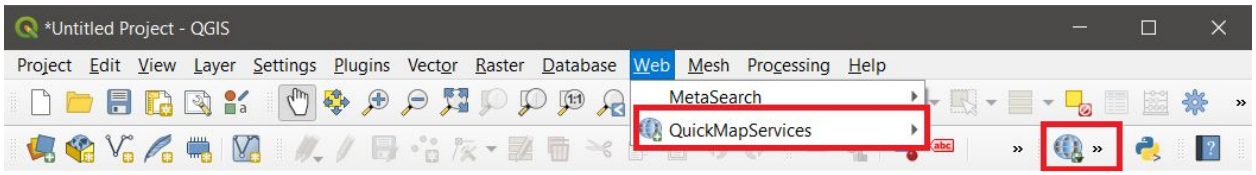
1. Using map services in QGIS via the *QuickMapServices* plugin.

Maps services provide topographic maps that can be used for various purposes. For example, it is useful to have a background topographic map as a visual reference for the other layers in QGIS. There are many different map service providers including Google and OpenStreetMap. To access various map services, we will use the *QuickMapServices* plugin for QGIS. To install this plugin, follow the steps below:

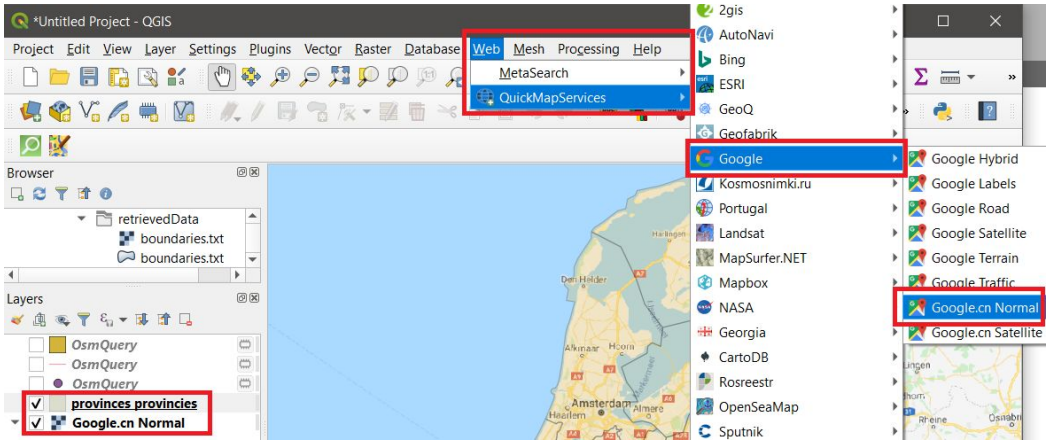
1. In the main menu of QGIS, go to "*Plugins -> Manage and Install Plugins ...*". This will open the *Plugins* dialogue window.
2. In the left side menu of the dialogue window, select the *All* option.
3. In the search field, type "*QuickMapServices*".
4. Below the search field, you should see the matching suggestions.
5. Select *QuickMapServices* from the suggestions and click the "*Install Plugin*" button.
6. Wait until the plugin is downloaded and installed and press the *Close* button.



Now, we are ready to add background map layers to the QGIS project. QuickMapServices can be accessed from either the main menu or the icon ribbon.



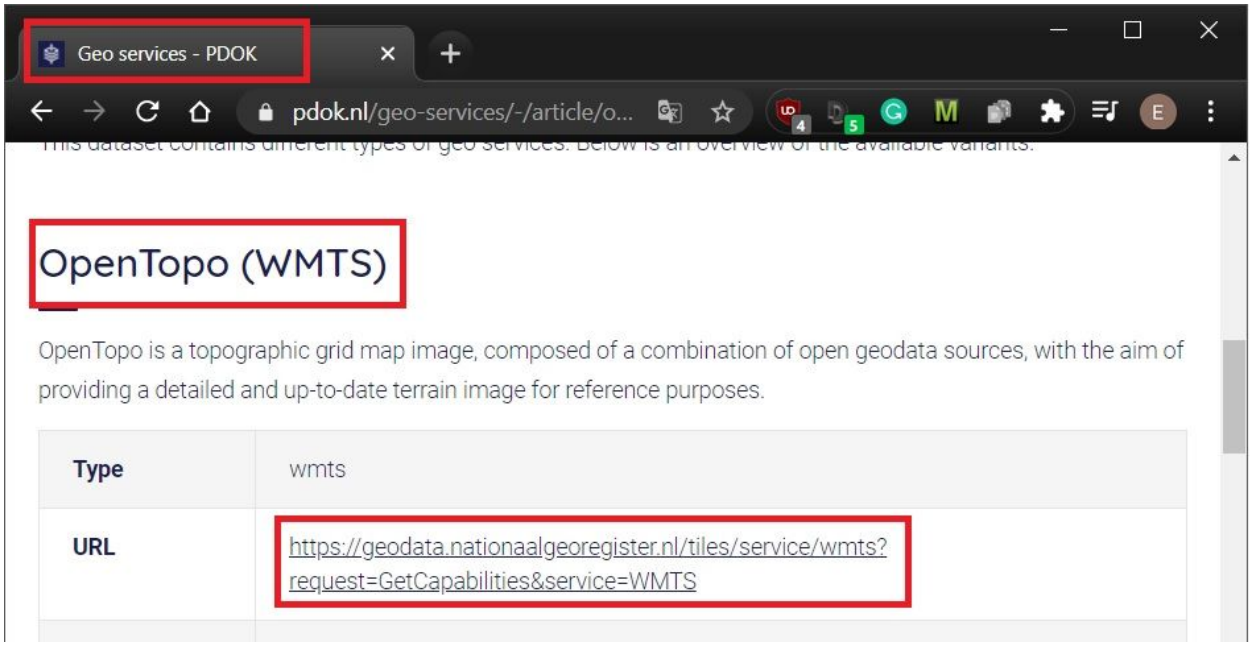
Let's add a map from Google as a background map for the layer of Dutch provinces (*provinces.gml*). In the main menu, go to "*Web -> QuickMapServices -> Google -> Google.cn Normal*". If you don't see the option for Google then go *"Web -> QuickMapServices -> Settings -> More service -> get contributed pack" and try again.* Afterwards, you should see a new layer "*Google.cn Normal*" added to the project.



The map of Dutch provinces overlaid over the map from Google should look like below. Note that, in the picture below, the opacity of the fill color for the provinces was decreased to make the map more interpretable. The opacity can be changed from the "*Layer Styling*" panel (*View -> Panels -> Layer Styling*). In a similar manner, you can try to add a map from OpenStreetMap which is also available via QuickMapServices.

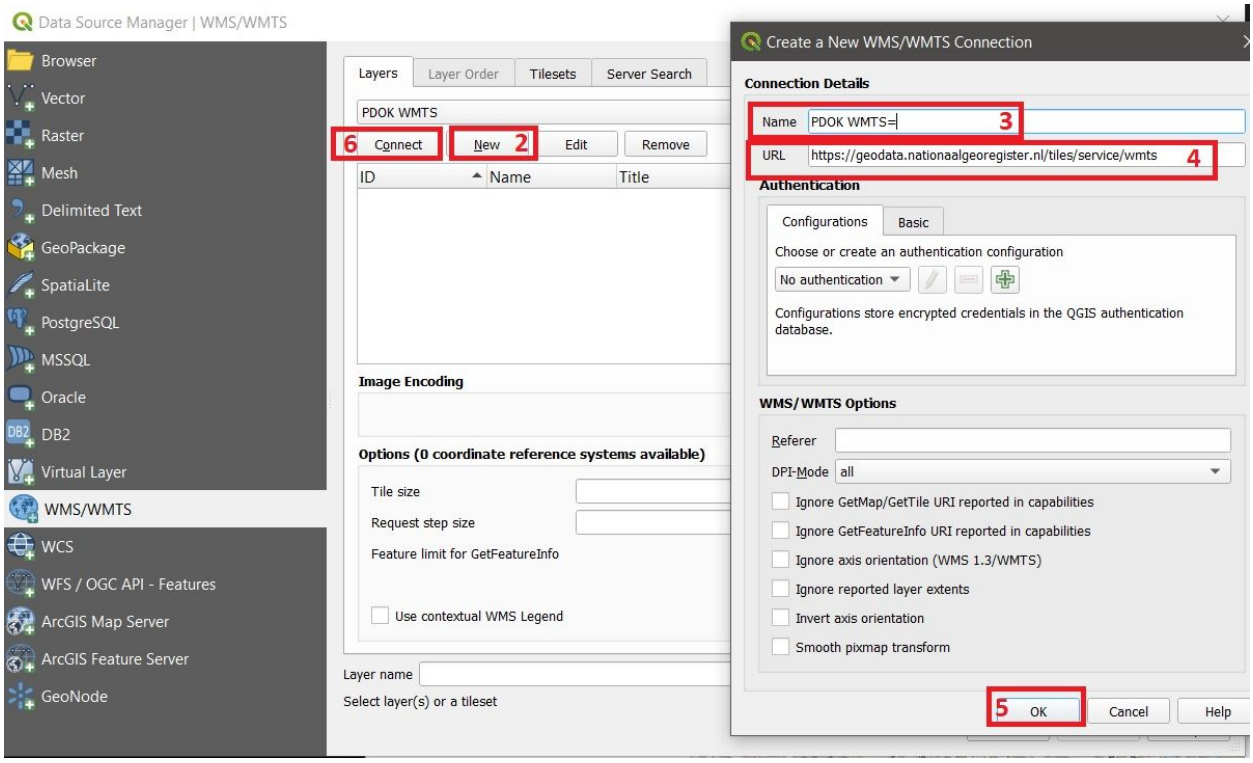
2. Using map services in QGIS via the built-in WMTS tool.

Not all map services are accessible via the QuickMapServices plugin. For example, PDOK also offers map services although it is not listed in QuickMapServices. Alternatively, QGIS has its built-in tool for connecting to any map service. While map services use a different protocol called WMTS (Web Map Tile Service), the working principles are very much similar to WFS. We need a unique web URL to connect to any WMTS server. For PDOK, this URL can be found at https://www.pdok.nl/geo-services/-/article/opentopo. If we go to this webpage, we can find the WMTS URL of the dataset as shown in the image below:

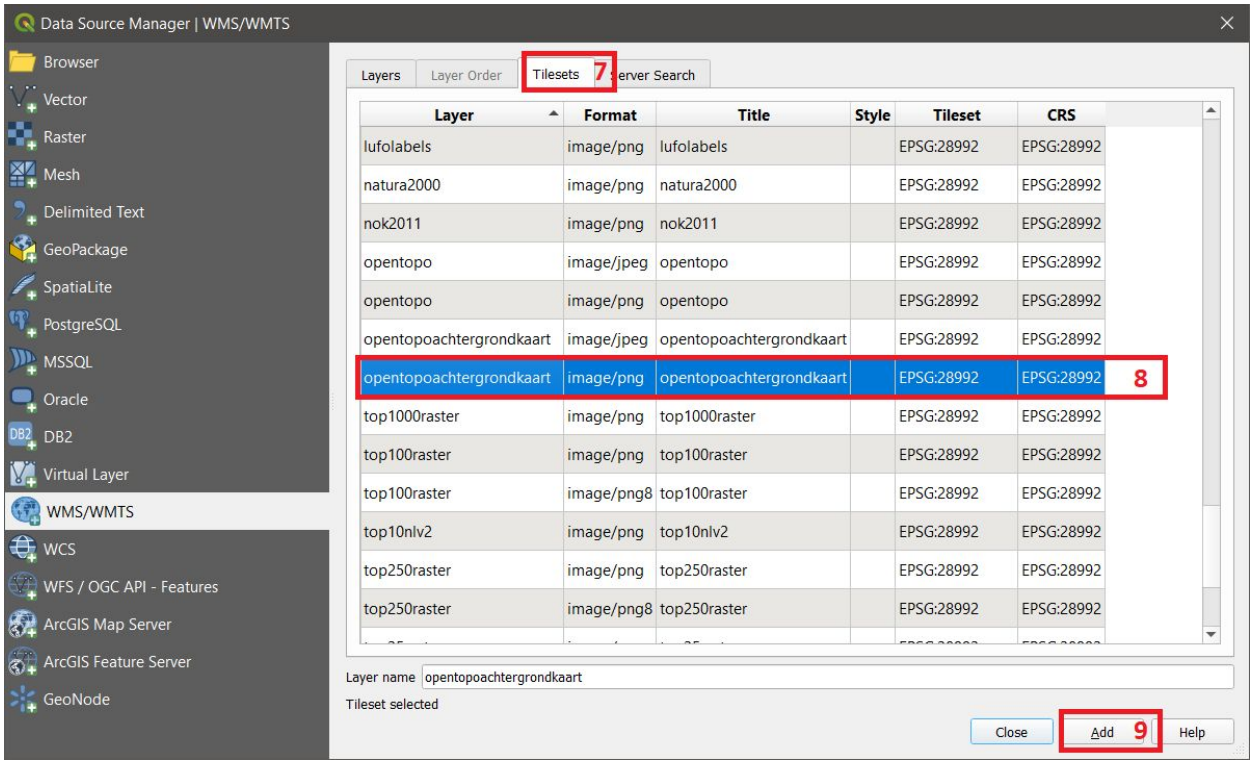


The URL should look like this:
https://geodata.nationaalgeoregister.nl/tiles/service/wmts?request=GetCapabilities&service=WMTS
Note that this is an HTTP query to the WMTS server. The actual WMTS URL is https://geodata.nationaalgeoregister.nl/tiles/service/wmts. Everything after the question mark "?" is a query string that is not necessary for the QGIS tool.

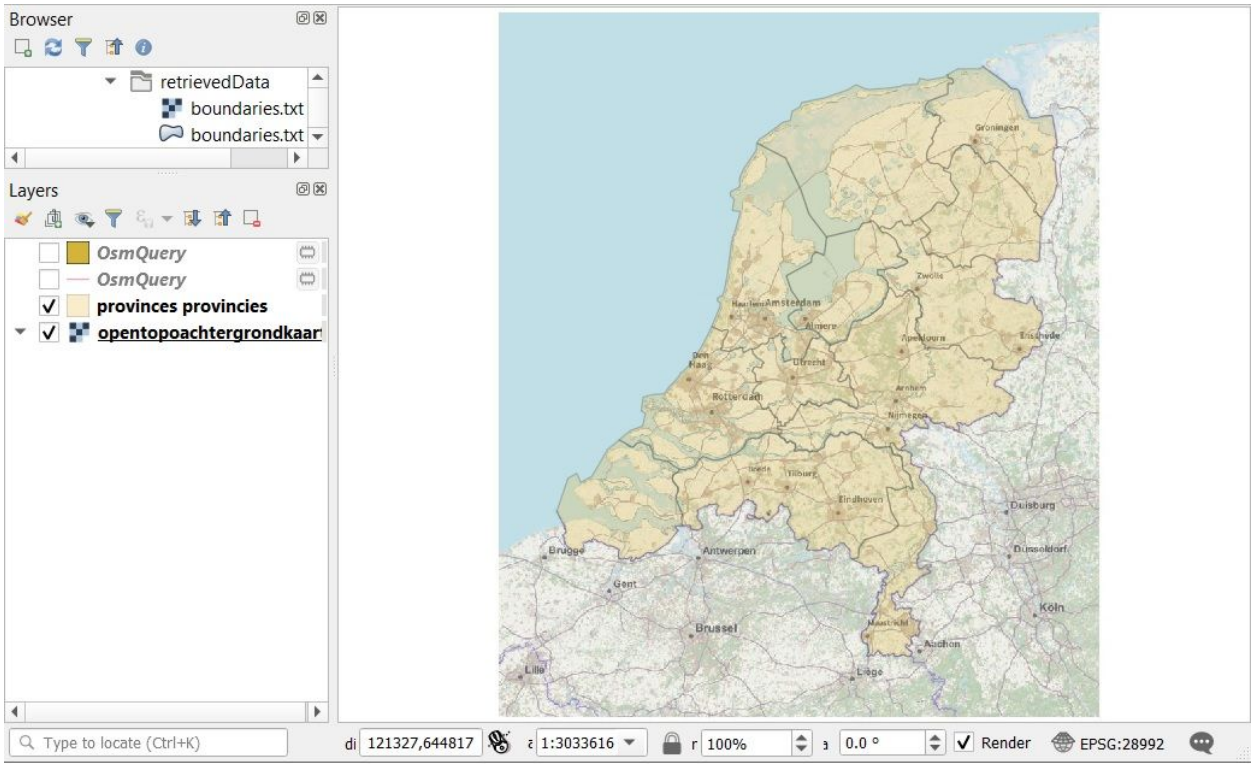
To connect to the PDOK WMTS server, follow the steps below:
1. In the main menu of QGIS, go to "*Layer -> Add Layer -> Add WMS/WMTS Layer ...*". This will open the "*Data Source Manager*" window.
2. In the "*Data Source Manager*" window, click the *New* button. This will open another window named "*Create a New WMS/WMTS Connection*".
3. Enter a name of your choice, for example, "*PDOK WMTS*".
4. Enter the WMTS URL. In this case, it is https://geodata.nationaalgeoregister.nl/tiles/service/wmts.
5. Click Ok.
6. In the "*Data Source Manager*" window, select "*PDOK WMTS*" and click the *Connect* button.



7. Upon a successful connection, the window will automatically switch into the *Tilesets* tab that lists the layers that can be downloaded from the WMTS server.
8. As an example, let's download a topographic map of the Netherlands specifically designed as a background map for data layers. Select the layer named "*opentopoachtergrondkaart*".
9. Click the *Add* button and then close the "*Data Source Manager*" window.



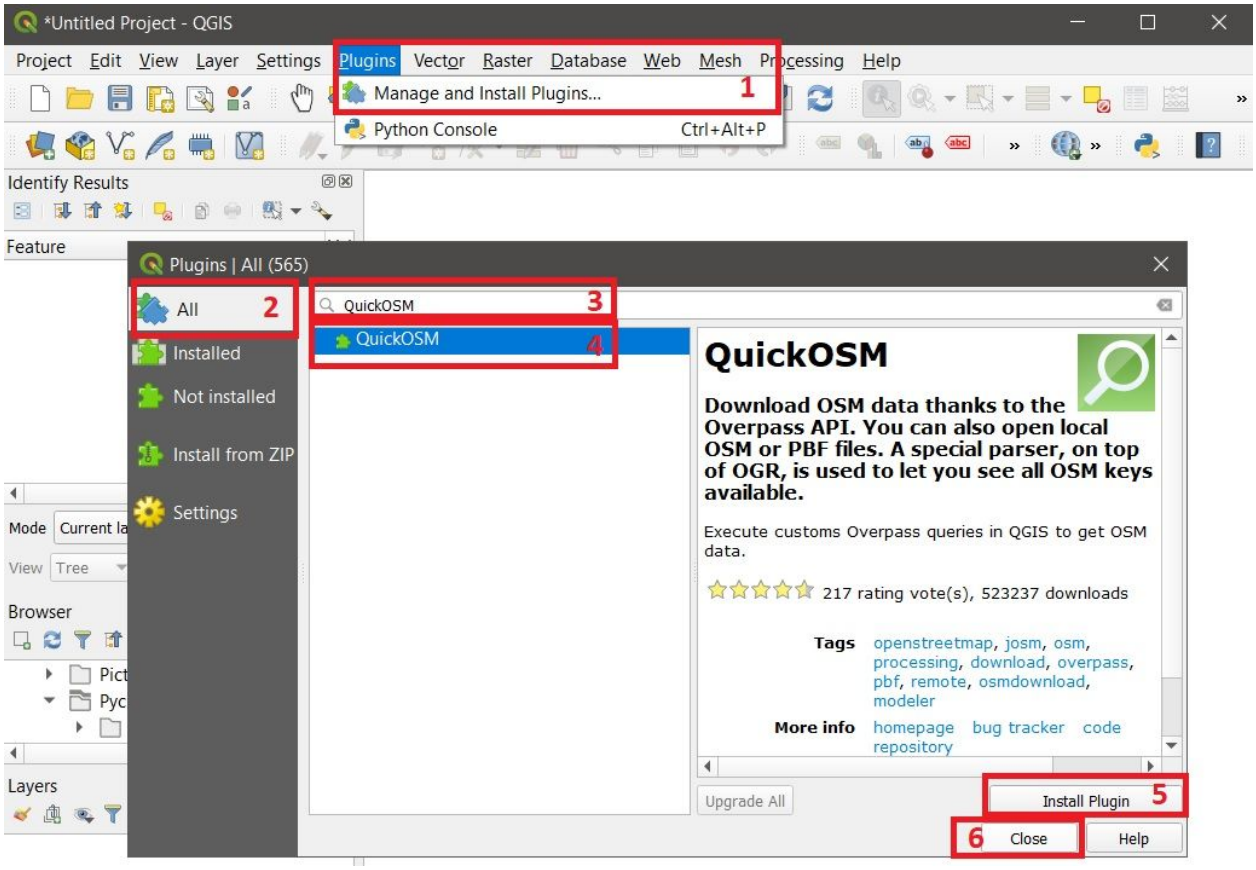
The resulting map of Dutch provinces with PDOK background map should look like this:



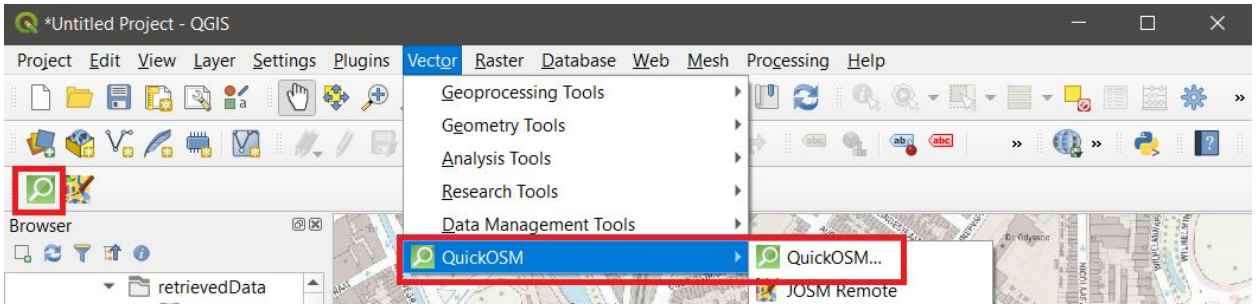
Retrieving OpenStreetMap (OSM) data with QGIS

OpenStreetMap (OSM) is arguably the largest open-source portal for spatial data. Therefore, it will be useful to know how to access this data within QGIS. For this purpose, we will use the *QuickOSM* plugin for QGIS. To install this plugin, follow the steps below:

1. In the main menu of QGIS, go to "*Plugins -> Manage and Install Plugins ...*". This will open the *Plugins* dialogue window.
2. In the left side menu of the dialogue window, select the *All* option.
3. In the search field, type "*QuickOSM*".
4. Below the search field, you should see the matching suggestions.
5. Select *QuickOSM* from the suggestions and click the "*Install Plugin*" button.
6. Wait until the plugin is downloaded and installed and press the *Close* button.

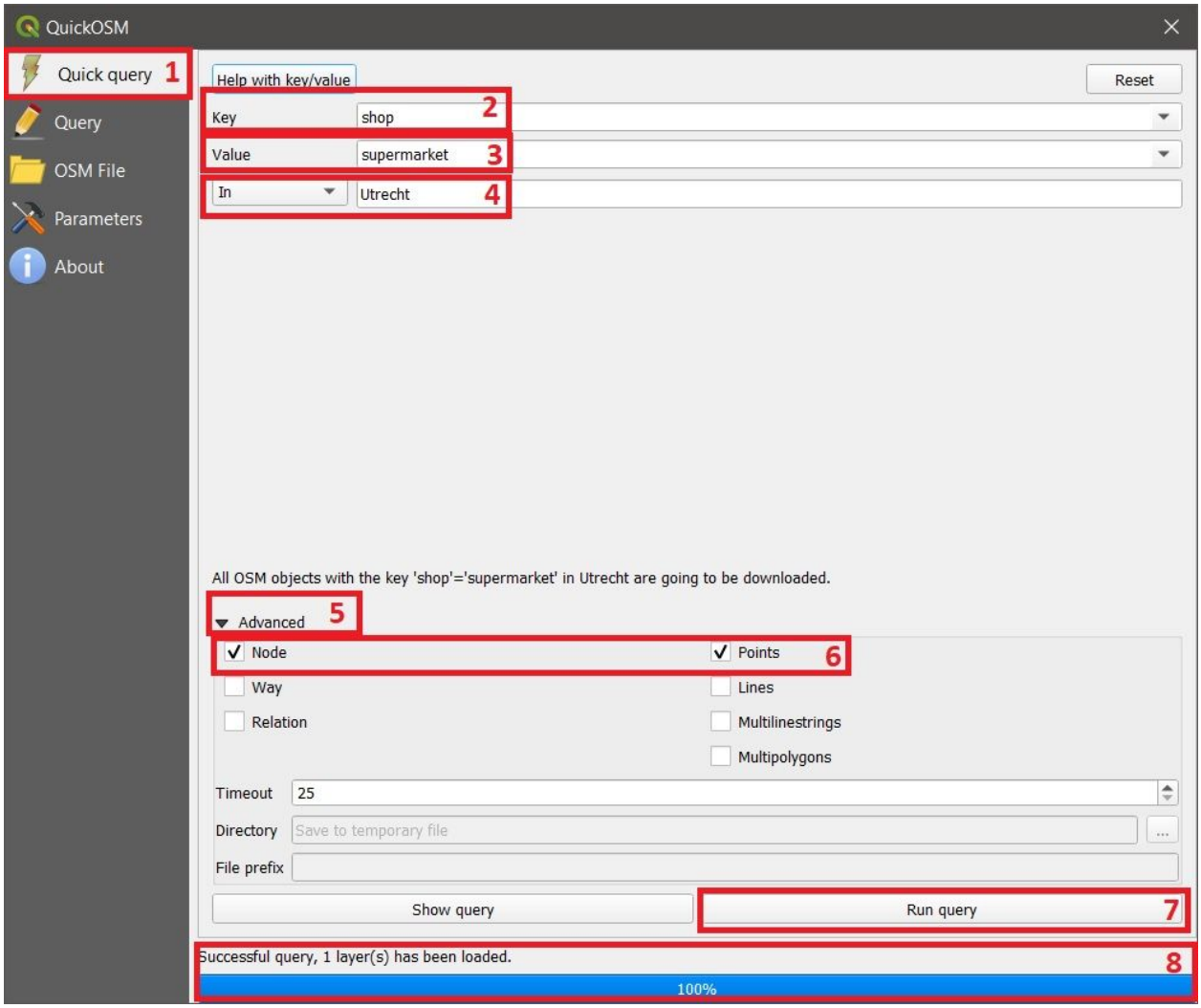


The plugin can be accessed through either the main menu or the icon ribbon.

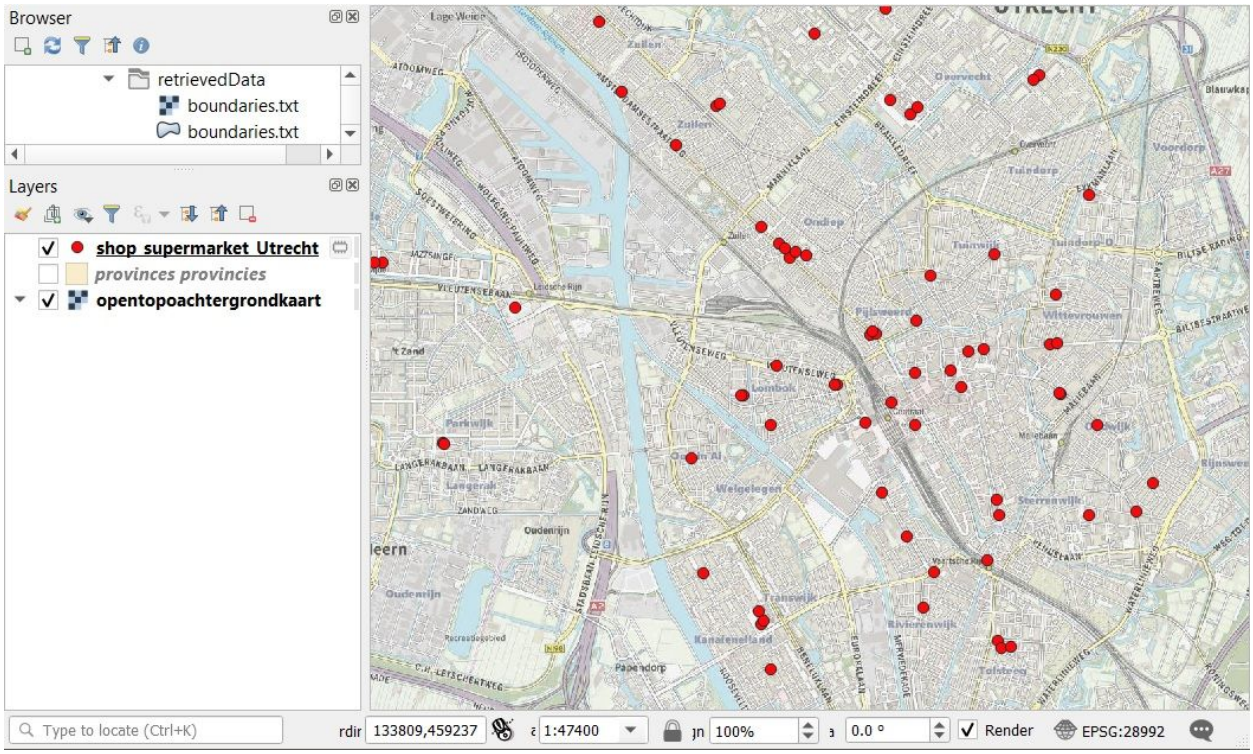


In OSM, any spatial feature (point, line, or polygon) is annotated with one or more key-value pairs. We can use these key-value pairs to search for the spatial features we are interested in. As a first example, lets find all supermarkets in Utrecht represented by point features. Open the plugin window and follow the steps below:

1. Select "*Quick query*" option in the left side menu.
2. In the *Key* field, enter or select from the drop-down menu the key *shop*.
3. In the *Value* field, enter the value *supermarket*.
4. In the *In* field, enter the extent of Utrecht.
5. Press the *Advanced* button to show the advanced options.
6. In the advanced options, select only *Node* and *Points*.
7. Press the "*Run query*" button to execute the query.
8. If any matching spatial features are found then they are automatically added as a layer to the current project.



The resulting map of supermarkets in Utrecht should look like the image below:

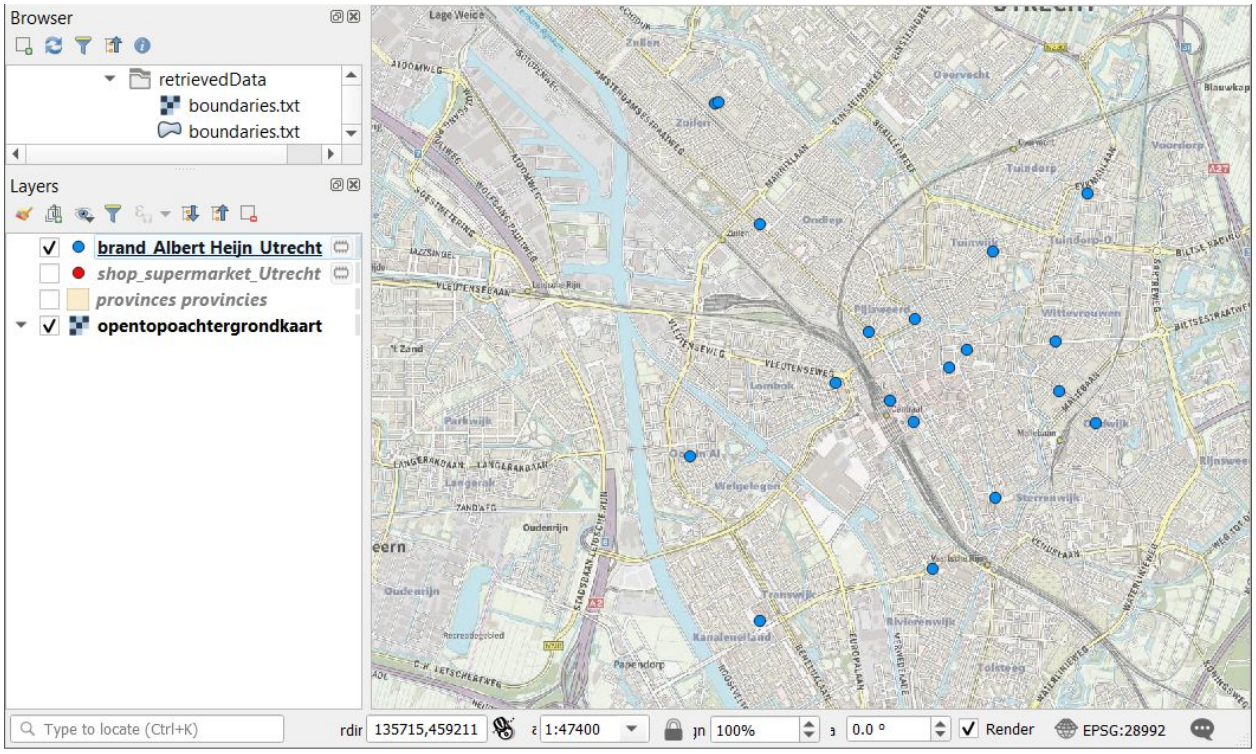


As was discussed earlier, each spatial feature can have more than one key-value pair. We can check these key-value pairs from the layer's attribute table. To open the attribute table, right click on the "*shop_supermarket_Utrecht*" layer and select the "*Open Attribute Table*" option from the popup menu. The attribute table should look like the one below:

shop_supermarket_Utrecht :: Features Total: 75, Filtered: 75, Selected: 0

	full_id	osm_id	osm_type	name	shop	addr:city	addr:housenumber	addr:postcode	addr:street	brand	brand:wikidata	brand:wikipedia
1	n188...	18811...	node	Albert Heijn	supermarket	Utrecht	37	3531BS	Damstraat	Albert Heijn	Q1653985	nl:Albert Heijn (...
2	n225...	22580...	node	PLUS	supermarket	Utrecht	63-65	3512AK	Voorstraat	PLUS	Q1978981	nl:PLUS (Nederl...
3	n164...	16408...	node	Albert Heijn	supermarket	Utrecht	367A	3551CK	Amsterdamsestraatweg	Albert Heijn	Q1653985	nl:Albert Heijn (...
4	n169...	16999...	node	Albert Heijn	supermarket	Utrecht	66G	3525AH	't Goylaan	Albert Heijn	Q1653985	nl:Albert Heijn (...

In the attribute table, each column name is a key, and each row contains key values for the same spatial feature. We can see the column named *shop*. This is the key we have used to retrieve this dataset. Another key is *brand* with brand names as values. We can also use this key-value pairs to retrieve, for example, only Albert Heijn supermarkets as shown in the map below:



Retrieving a remote GeoJSON dataset with QGIS

GeoJSON is an increasingly popular open format for sharing spatial data. It is based on JSON, a very popular format for exchanging web-based data. Many data portal share direct link to datasets in GeoJSON format. For example, Amsterdam open geo-data portal (https://maps.amsterdam.nl/open_geodata) supports GeoJSON for all its datasets. We will import one of its datasets into QGIS using the built-in tool. The dataset is about public green spaces and park in Amsterdam (https://maps.amsterdam.nl/open_geodata/?k=99). The picture below shows the dataset's webpage and the link to the distribution in GeoJSON format:

Layer	**PARKPLANTSOENGROEN**
Page link	https://maps.amsterdam.nl/open_geodata/?k=99
Attributes	Stadsdeel Naam Stadspark Oppervlakte_m2
Number of objects	122
Recency	april 2014
Read the Explanation at	**https://maps.amsterdam.nl/stadsparken/ >**
Extra explanation	
Sourceholder	Gemeente Amsterdam - Ruimte en Duurzaamheid
Contact persoon	Geertje Wijten
Email	g.wijten@amsterdam.nl

You agree to the **Terms of Use >**

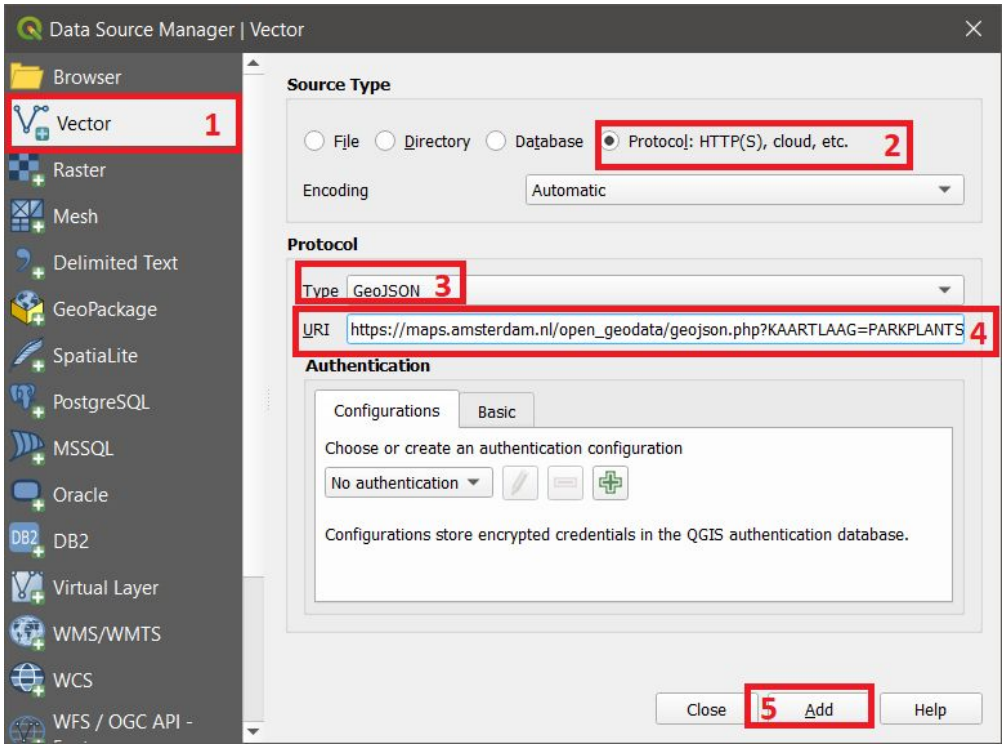
Download: **.csv (Excel) >**

Download: **.json (GeoJSON) >**

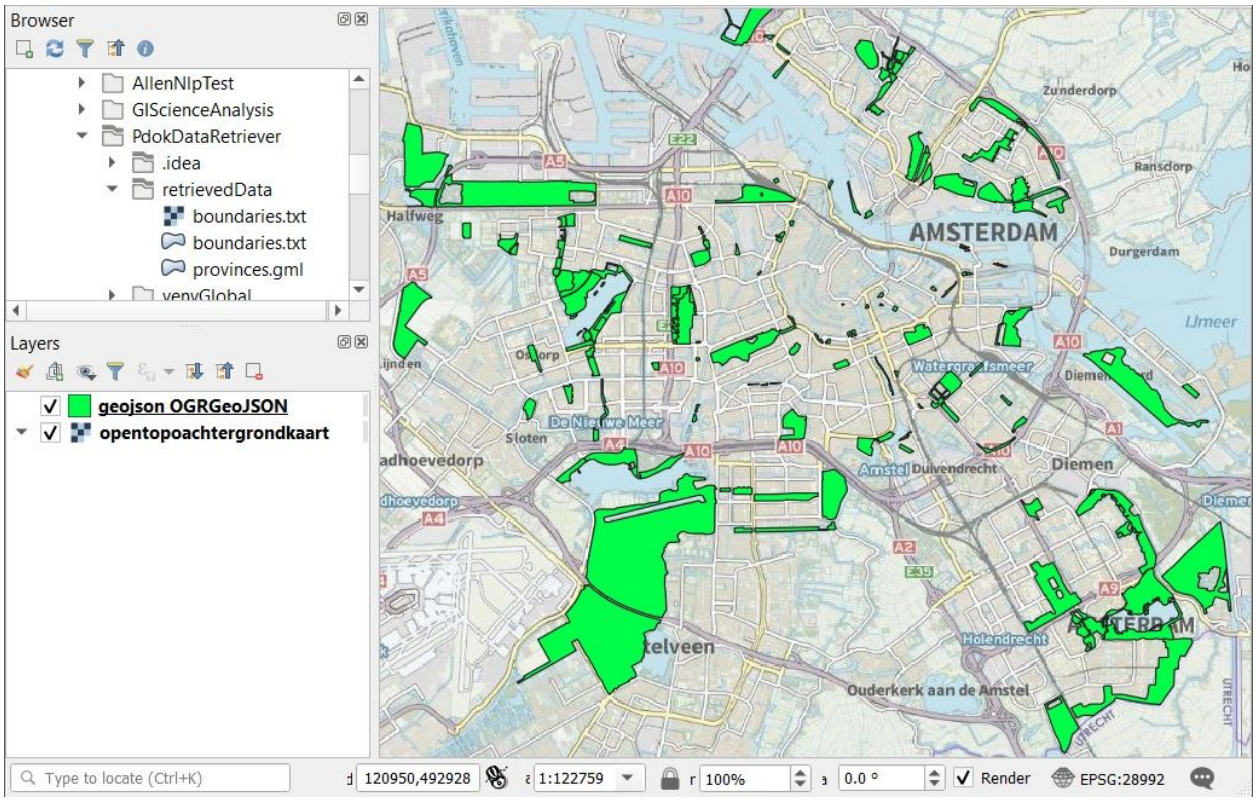
Download: **.mif (GIS) > and .mid (GIS) >**

To import the dataset, follow the steps below:

1. In the main menu, go to "*Layer -> Add Layer -> Add Vector Layer ...*" to open the "*Data Source Manager*".
2. In "*Source Type*", select "*Protocol: HTTP(S), ...*".
3. In *Protocol Type*, select *GeoJSON*.
4. In *Protocol URI*, enter the URL of the dataset's GeoJSON distribution.
5. Click the *Add* button and close the "*Data Source Manager*" window.



The dataset should be imported as a layer to your current project. The resulting map should look like the image below. The green polygons are from the dataset and depict the green spaces and parks of Amsterdam.



Creating a standalone Python application using QGIS libraries in PyCharm

In this section, we will learn how to create a standalone Python application that makes use of the QGIS libraries. The application will download a remote dataset, make a layer out of it, and visualize the layer in a map canvas.

At first, let's create a new PyCharm project that we will use to connect to QGIS Python library:
1. Close all currently open PyCharm projects by selecting "*File -> Close Project*".
2. Create a new PyCharm project called "*PyQGIS*".
3. Once the project is created and initialized, close the project "*PyQGIS*" and close PyCharm.

Now, we have to set environment variables so that PyCharm knows where to find and import QGIS Python libraries and invoke QGIS virtual environments. For this purpose, we will create a Windows batch file that will list all environment variables pointing to the libraries and QGIS executables. This batch file will also automatically start PyCharm that is connected to the QGIS environment. Follows steps below to create the batch file:

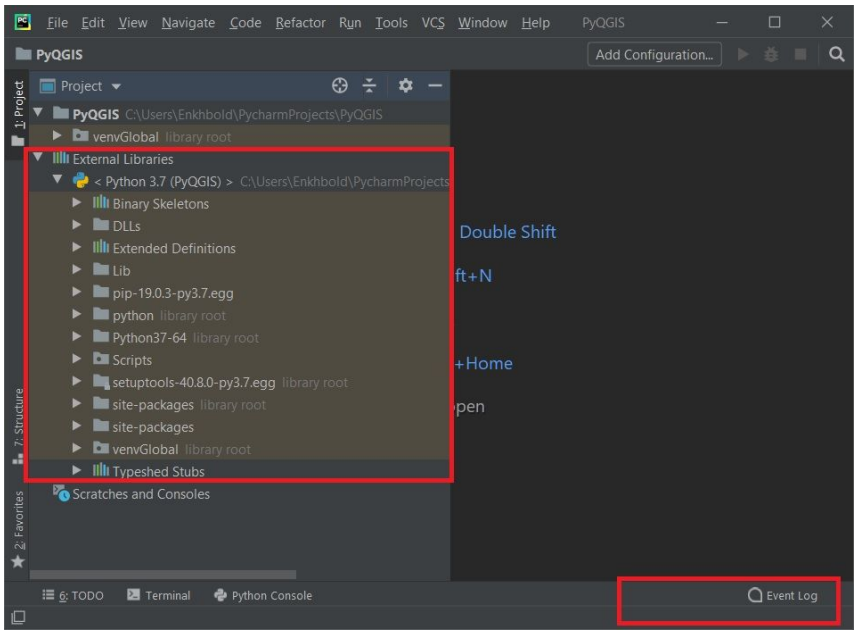
1. At the location of your choice, create a new file and name it "*pyqgis.bat*". Note that the file extension should be "bat" and not anything else.
2. Open the "*pyqgis.bat*" with a text editor (e.g., notepad or notepad++).
3. Paste to "*pyqgis.bat*" the script below or from "*pycharmQGIS-Template.bat*" (on blackboard):

```
SET QGIS_ROOT=<path to the root folder of QGIS installation>
SET PYCHARM="<path to the pycharm.bat file in Pycharm installation folder>"

call "%QGIS_ROOT%"\bin\o4w_env.bat
call "%QGIS_ROOT%"\apps\qt5\bin\qtenv2.bat

path %PATH%;%QGIS_ROOT%\apps\qgis\bin
path %PATH%;%QGIS_ROOT%\apps\qt5\bin

set QT_PLUGIN_PATH=%QGIS_ROOT%\apps\qgis\qtplugins;%QGIS_ROOT%\apps\qt5\plugins

set PYTHONPATH=%PYTHONPATH%;%QGIS_ROOT%\apps\qgis\python
set PYTHONPATH=%PYTHONPATH%;%QGIS_ROOT%\apps\Python37\Lib\site-packages

set QGIS_PREFIX_PATH=%QGIS_ROOT%\apps\qgis


start "PyCharm aware of QGIS" /B %PYCHARM% %*
```

4. In "*pyqgis.bat*", make sure to set the paths for `QGIS_ROOT` and `PYCHARM` variables. `QGIS_ROOT` should point to the root folder where QGIS was installed. `PYCHARM` should point to the pycharm.bat file inside PyCharm installation folder. Below is the example how these variable can look like:

```
SET QGIS_ROOT=C:\InstalledPrograms\QGIS 3.12
SET PYCHARM="C:\InstalledPrograms\JetBrains\PyCharm Community Edition
2019.2.2\bin\pycharm.bat"
```

5. The next set of environment variables point to the libraries and executables inside QGIS folder. Since the exact paths may differ among different versions of QGIS, make sure these paths are correct for the version of QGIS you have. Once you have verified all paths, save and close the batch file.
6. Execute the "*pyqgis.bat*" batch file. This will open a Windows command line window where all batch scripts are executed. The command line will start PyCharm. By default, PyCharm should automatically open the "*PyQGIS*" project. If it does not, manually open the "*PyQGIS*" project. In the project, PyCharm will start automatically importing and loading QGIS libraries. The loading progress can be seen at the bottom-right corner of the PyCharm window. The newly imported libraries should show up on the left-hand side *Project* pane under the "*External Libraries -> <Python 3.7 (PyQGIS)>*" option.



7. Now, we are ready to do some coding. Inside the "*PyQGIS*" project, create a new Python file named, for example, "*PyQGISApp*".
8. Copy/paste code from the file *"PyQGISApp.py"* (on blackboard) to the newly created file.

Let's go through the parts of the code necessary for any standalone Python application that uses QGIS libraries.

NOTE: With the code below, you will get syntax errors in the PyCharm editor. You can ignore these errors. The errors are caused by libraries that are dynamically loaded at runtime and, therefore, cannot be identified by the editor.

NOTE: When running the code, If you get an error message related to importing DLL libraries then (1) carefully check if ALL the paths in the batch file are correct and (2) ensure that you opened PyCharm by running the batch file.

First, there are import statements that refer to various QGIS modules.

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

Next, we should indicate the path to the location of QGIS. Note that the path is NOT the path to the root folder where QGIS was installed but rather path to the folder `apps/qgis` inside the root folder.

```
QgsApplication.setPrefixPath("C:/InstalledPrograms/QGIS 3.12/apps/qgis", True)
```

Afterwards, we need to initialize the QGIS application environment:

```
qgs = QgsApplication([], True)
qgs.initQgis()
```

After the initialization, any custom code can be written that makes use of the QGIS or other libraries. Finally, the three lines of code below close the QGIS application environment. These methods should be called at the very end so that any custom code can be executed beforehand.

```
exitcode = qgs.exec_()
qgs.exitQgis()
sys.exit(exitcode)
```

The custom code consists of three parts: (1) download remote dataset via WFS and create a layer; (2) register a map layer; (3) visualize the map layer.
The code below retrieves a dataset of regions of Italy from a WFS server and creates with it an instance of `QgsVectorLayer` class. The three parameters are (1) HTTP request to the WFS server, (2) a custom name for the vector layer, and (3) a data retrieval format which is *WFS* in this particular example.

```
uri =
"https://demo.geo-solutions.it/geoserver/ows?service=WFS&version=1.1.0&request=GetFe
ature&typename=geosolutions:regioni"

vlayer = QgsVectorLayer(uri, "Regions of Italy", "WFS")
```

Next, we register and add the new map layer to a virtual QGIS project:

```
QgsProject.instance().addMapLayer(vlayer)
```

To visualize the map layer, at first, we need to prepare a canvas for drawing a map. The code below creates a canvas window, sets the window title, sets the background color to white, enables anti-aliasing, and displays the window.

```
# create a canvas object
canvas = QgsMapCanvas()
# set window title
canvas.setWindowTitle("Map window")
# set the background color for the canvas
canvas.setCanvasColor(Qt.white)
# enable antialiasing in the canvas
canvas.enableAntiAliasing(True)
# show the canvas
canvas.show()
```

The final steps are to add the map layer to the canvas and match the canvas size to the layer's extent.

```
# set the map canvas layer set
canvas.setLayers([vlayer])
# set extent to the extent of our layer
canvas.setExtent(vlayer.extent())
canvas.zoomToFullExtent()
```

The map visualization should look like the image below:

Assignment: searching for, downloading, and interpreting datasets relevant for a geo-analytic scenario

Further in the course, you will work on a concrete case study. One example is about assessing the suitability of Amsterdam for active transport (walking/biking). For the suitability assessment, you may need to consider following six factors:

- Greenness of the city (e.g., parks, trees, green areas)
- Landuse types in the city
- Digital Elevation Map of the city
- Points of interest in the city (e.g, schools, shops, hospital, etc)
- Residential buildings in the city
- Street network in the city

In this assignment, you need to search for and download **three datasets** that may represent some of the above factors. You should use QGIS to download and visualize the datasets. You can download from any data portal of your choice.

For this assignment, you need to submit a Word document where you describe each dataset you have downloaded. The description should include:

- The source of the dataset (from where you downloaded the dataset)
- Download link (if available)
- A short description of the dataset content (what the dataset is about)
- A screenshot of QGIS visualization of the dataset
- Spatial metadata description that includes:
    - Number of dimensions (1D, 2D, 3D, 4D) in the dataset
    - Spatial representation type (e.g. raster, vector, text table) in the dataset
    - Coordinate Reference System(s) of the dataset

Optional exercises

These exercises are optional, and you don't have to submit anything for these exercises.

Exercise 1: A standalone Python app for retrieving and visualizing a PDOK dataset.

In this exercise, you need to create a standalone Python app that

1. downloads a PDOK dataset using the OWSLIB library and stores the dataset locally,
2. loads and visualizes the stored dataset using QGIS libraries.

Essentially, you need to combine codes from the PdokDataRetriever and PyQGIS projects into a single Python application.

Exercise 2: Finding a correct CRS for a regional dataset.

In this exercise, you need to find and download a spatial dataset belonging to regions OUTSIDE of Europe and UK (e.g., Asia, Australia, Africa, ...). You need to import and visualize the dataset in QGIS while making sure that a correct CRS is being used for the dataset.

Exercise 3: Handling big spatial data.

Spatial datasets can be very large, and it is often the case that WFS servers have limits on the size of the data that can be downloaded with a single request. To address this issue, WFS supports *pagination*. With pagination, it is possible to divide a large dataset into chunks and retrieve one chunk per a request. You can explore the WFS references to learn how to do pagination:

- PDOK WFS reference: https://pdok-ngr.readthedocs.io/handleidingen.html#wfs-pagination
- Official WFS reference: https://docs.geoserver.org/latest/en/user/services/wfs/index.html
- OWSLib WFS reference: https://geopython.github.io/OWSLib/#wfs

In this exercise, you need to write a Python application that uses pagination with OWSLib to download a PDOK dataset of your choice. The dataset should be divided into, at least, three chunks.

References

A reference of the parameters supported by the Web Feature Service (WFS) standard. https://docs.geoserver.org/latest/en/user/services/wfs/reference.html

A brief description of PDOK's WFS. https://www.pdok.nl/wfs

Short introduction to how to use PDOK's WFS. Text is in Dutch. https://pdok-ngr.readthedocs.io/services.html#web-feature-service-wfs

Manuals on how to control output pagination, output formats, and output coordinates in PDOK's WFS. Text is in Dutch. https://pdok-ngr.readthedocs.io/handleidingen.html#wfs-pagination

Example code of using Python's OWSLib library to connect to and use WFS. https://geopython.github.io/OWSLib/#wfs

PDOK datasets. https://www.pdok.nl/datasets

Summary of PDOK service and APIs. https://www.pdok.nl/services-en-api-s

PDOK tutorial for using QGIS with WFS. Text is in Dutch. https://pdok-ngr.readthedocs.io/downloaden.html#op-de-desktop-qgis

QGIS tutorial for using WFS. https://docs.qgis.org/3.10/en/docs/training_manual/online_resources/wfs.html