

Find Neighbor Polygons in a Layer

QGIS Tutorials and Tips



Author

Ujaval Gandhi

<http://google.com/+UjavalGandhi>

Translations by

Marina Pavlova Ilya Trofimov Fayçal Fatihi

Introduction to QGIS and Python

QGIS is a free and open-source geographic information system (GIS) for desktop use. It is a powerful tool for working with spatial data, and it is widely used by researchers, students, and professionals alike. QGIS is built on a modular architecture, which allows it to be extended with a variety of plugins. One of the most powerful ways to extend QGIS is by using Python. Python is a high-level, interpreted programming language that is easy to learn and use. It is also a very powerful language, and it can be used to perform a wide range of tasks, from simple data manipulation to complex spatial analysis. In this tutorial, we will explore the basics of QGIS and Python, and we will see how they can be used together to create powerful GIS applications.

Getting started with QGIS

QGIS is a cross-platform application, which means it can be run on a variety of operating systems, including Windows, macOS, and Linux. To get started with QGIS, you first need to download and install the software. Once you have installed QGIS, you can open the application and start working with your data. QGIS has a user-friendly interface, and it includes a variety of tools for working with spatial data. In this section, we will explore the basics of the QGIS interface, and we will see how to load and display spatial data.

Working with vector data

QGIS can work with a variety of data formats, but it is most commonly used to work with vector data. Vector data is a type of spatial data that is represented by points, lines, and polygons. It is a very flexible format, and it can be used to represent a wide range of features, from individual points to complex polygons. In QGIS, vector data is typically stored in a file format called **Admin 0 - Countries** from Natural Earth. This file contains data about the world's countries, including their names, boundaries, and other attributes. We can load this data into QGIS and use it to create maps and perform spatial analysis.

Using Python to automate tasks

One of the most powerful features of QGIS is its ability to be extended with Python. Python is a high-level programming language that is easy to learn and use. It is also a very powerful language, and it can be used to perform a wide range of tasks, from simple data manipulation to complex spatial analysis. In QGIS, Python is used to create **neighbors.py** scripts that can be used to automate tasks, such as loading data, performing analysis, and creating maps.

Conclusion

1. Load the **ne_10m_admin_0_countries** layer into QGIS. Go to **Layer** > **Add** **Vector Layer**.



3. ■■■■■■■■■■ ■■■■■■ Plugins ■ Python Console.



6. `ne_10m_admin_0_countries`, `ne_10m_admin_0_countries`, `ne_10m_admin_0_countries`
`_NAME_FIELD` `_SUM_FIELD` `ne_10m_admin_0_countries`
`ne_10m_admin_0_countries`, `ne_10m_admin_0_countries`, `ne_10m_admin_0_countries`
`ne_10m_admin_0_countries`: `guilabel: save` `ne_10m_admin_0_countries`: `guilabel: Editor`, `ne_10m_admin_0_countries`
`ne_10m_admin_0_countries`: `guilabel: Run script`, `ne_10m_admin_0_countries`



7. **ne_10m_admin_0_countries** **Open** **Table**.

Attribute table - ne_10m_admin_0_countries :: Features total: 255, filtered: 255, selected: 0

| | ION | REGION_WB | NAME_LEN | LONG_LEN | ABBREV_LEN | TINY | HOMEPART | NEIGHBORS | SUM |
|----|---------|---------------------|----------|----------|------------|--------|----------|----------------------|------------|
| 0 | | Latin America & ... | 5.00 | 5.00 | 5.00 | 4.00 | -99.00 | NULL | 0 |
| 1 | sia | South Asia | 11.00 | 11.00 | 4.00 | -99.00 | 1.00 | Iran,Turkmenista... | 1621125240 |
| 2 | a | Sub-Saharan Africa | 6.00 | 6.00 | 4.00 | -99.00 | 1.00 | Namibia,Zambia,... | 86676756 |
| 3 | | Latin America & ... | 8.00 | 8.00 | 4.00 | -99.00 | -99.00 | NULL | 0 |
| 4 | urope | Europe & Central... | 7.00 | 7.00 | 4.00 | -99.00 | 1.00 | Macedonia,Greec... | 15281164 |
| 5 | urope | Europe & Central... | 5.00 | 13.00 | 5.00 | 5.00 | -99.00 | NULL | 0 |
| 6 | urope | Europe & Central... | 7.00 | 7.00 | 4.00 | 5.00 | 1.00 | France,Spain | 104582794 |
| 7 | ia | Middle East & No... | 20.00 | 20.00 | 6.00 | -99.00 | 1.00 | Saudi Arabia,Oman | 32104718 |
| 8 | ica | Latin America & ... | 9.00 | 9.00 | 4.00 | -99.00 | 1.00 | Bolivia,Paraguay,... | 235606259 |
| 9 | ia | Europe & Central... | 7.00 | 7.00 | 4.00 | -99.00 | 1.00 | Georgia,Turkey,I... | 156089287 |
| 10 | | East Asia & Pacific | 14.00 | 14.00 | 9.00 | 3.00 | -99.00 | NULL | 0 |
| 11 | | Antarctica | 10.00 | 10.00 | 4.00 | -99.00 | 1.00 | NULL | 0 |
| 12 | d Ne... | East Asia & Pacific | 23.00 | 27.00 | 7.00 | -99.00 | -99.00 | NULL | 0 |
| 13 | ope... | Sub-Saharan Africa | 22.00 | 35.00 | 10.00 | 2.00 | -99.00 | NULL | 0 |
| 14 | | Latin America & ... | 17.00 | 19.00 | 6.00 | 4.00 | 1.00 | NULL | 0 |
| 15 | d Ne... | East Asia & Pacific | 9.00 | 9.00 | 4.00 | -99.00 | 1.00 | NULL | 0 |
| 16 | rope | Europe & Central... | 7.00 | 7.00 | 5.00 | -99.00 | 1.00 | Italy,Hungary,Slo... | 175681436 |
| 17 | ia | Europe & Central... | 10.00 | 10.00 | 4.00 | -99.00 | 1.00 | Georgia,Turkey,R... | 290858866 |
| 18 | ica | Sub-Saharan Africa | 7.00 | 7.00 | 4.00 | -99.00 | 1.00 | Rwanda,Tanzani... | 120214356 |
| 19 | rope | Europe & Central... | 7.00 | 7.00 | 5.00 | -99.00 | 1.00 | France,Netherla... | 163595324 |
| 20 | rica | Sub-Saharan Africa | 5.00 | 5.00 | 5.00 | -99.00 | 1.00 | Nigeria,Niger,Bur... | 186301451 |
| 21 | rica | Sub-Saharan Africa | 12.00 | 12.00 | 4.00 | -99.00 | 1.00 | Mali,Niger,Ghana... | 87234511 |
| 22 | sia | South Asia | 10.00 | 10.00 | 5.00 | -99.00 | 1.00 | India,Myanmar | 1214216958 |

Show All Features

#####

```
#####
# Copyright 2014 Ujaval Gandhi
#
#This program is free software; you can redistribute it and/or
#modify it under the terms of the GNU General Public License
#as published by the Free Software Foundation; either version 2
#of the License, or (at your option) any later version.
#
#This program is distributed in the hope that it will be useful,
#but WITHOUT ANY WARRANTY; without even the implied warranty of
#MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
#GNU General Public License for more details.
#
#You should have received a copy of the GNU General Public License
#along with this program; if not, write to the Free Software
#Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.
#
#####
from qgis.utils import iface
from PyQt4.QtCore import QVariant

# Replace the values below with values from your layer.
# For example, if your identifier field is called 'XYZ', then change the line
```

```

# below to _NAME_FIELD = 'XYZ'
_NAME_FIELD = 'NAME'
# Replace the value below with the field name that you want to sum up.
# For example, if the # field that you want to sum up is called 'VALUES', then
# change the line below to _SUM_FIELD = 'VALUES'
_SUM_FIELD = 'POP_EST'

# Names of the new fields to be added to the layer
_NEW_NEIGHBORS_FIELD = 'NEIGHBORS'
_NEW_SUM_FIELD = 'SUM'

layer = iface.activeLayer()

# Create 2 new fields in the layer that will hold the list of neighbors and sum
# of the chosen field.
layer.startEditing()
layer.dataProvider().addAttributes(
    [QgsField(_NEW_NEIGHBORS_FIELD, QVariant.String),
     QgsField(_NEW_SUM_FIELD, QVariant.Int)])
layer.updateFields()
# Create a dictionary of all features
feature_dict = {f.id(): f for f in layer.getFeatures()}

# Build a spatial index
index = QgsSpatialIndex()
for f in feature_dict.values():
    index.insertFeature(f)

# Loop through all features and find features that touch each feature
for f in feature_dict.values():
    print 'Working on %s' % f[_NAME_FIELD]
    geom = f.geometry()
    # Find all features that intersect the bounding box of the current feature.
    # We use spatial index to find the features intersecting the bounding box
    # of the current feature. This will narrow down the features that we need
    # to check neighboring features.
    intersecting_ids = index.intersects(geom.boundingBox())
    # Initialize neighbors list and sum
    neighbors = []
    neighbors_sum = 0
    for intersecting_id in intersecting_ids:
        # Look up the feature from the dictionary
        intersecting_f = feature_dict[intersecting_id]

        # For our purpose we consider a feature as 'neighbor' if it touches or
        # intersects a feature. We use the 'disjoint' predicate to satisfy
        # these conditions. So if a feature is not disjoint, it is a neighbor.
        if (f != intersecting_f and
            not intersecting_f.geometry().disjoint(geom)):
            neighbors.append(intersecting_f[_NAME_FIELD])
            neighbors_sum += intersecting_f[_SUM_FIELD]
    f[_NEW_NEIGHBORS_FIELD] = ','.join(neighbors)
    f[_NEW_SUM_FIELD] = neighbors_sum
    # Update the layer with new attribute values.
    layer.updateFeature(f)

```

```
layer.commitChanges()  
print 'Processing complete.'
```