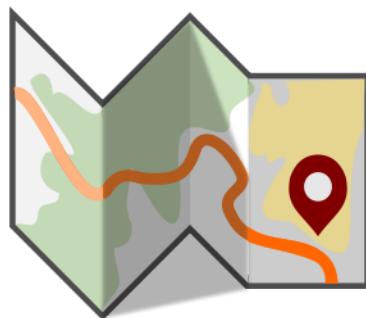


Writing Python Scripts for Processing Framework (QGIS3)

QGIS Tutorials and Tips



Author

Ujaval Gandhi

<http://www.spatialthoughts.com>

Writing Python Scripts for Processing Framework (QGIS3)

One can write standalone pyqgis scripts that can be run via the Python Console in QGIS. With a few tweaks, you can make your standalone scripts run via the Processing Framework. This has several advantages. First, taking user input and writing output files is far easier because Processing Framework offers standardized user interface for these. Second, having your script in the Processing Toolbox also allows it to be part of any Processing Model or be run as a Batch job with multiple inputs. This tutorial will show how to write a custom python script that can be part of the Processing Framework in QGIS.

Note

The Processing API was completely overhauled in QGIS3. Please refer to [this guide](#) for best practices and tips.

Overview of the task

Our script will perform a dissolve operation based on a field chosen by the user. It will also sum up values of another field for the dissolved features. In the example, we will dissolve a world shapefile based on a `CONTINENT` attribute and sum up `POP_EST` field to calculate total population in the dissolved region.

Get the data

We will use the [Admin 0 - Countries](#) dataset from Natural Earth.

Download the [Admin 0 - countries](#) shapefile..

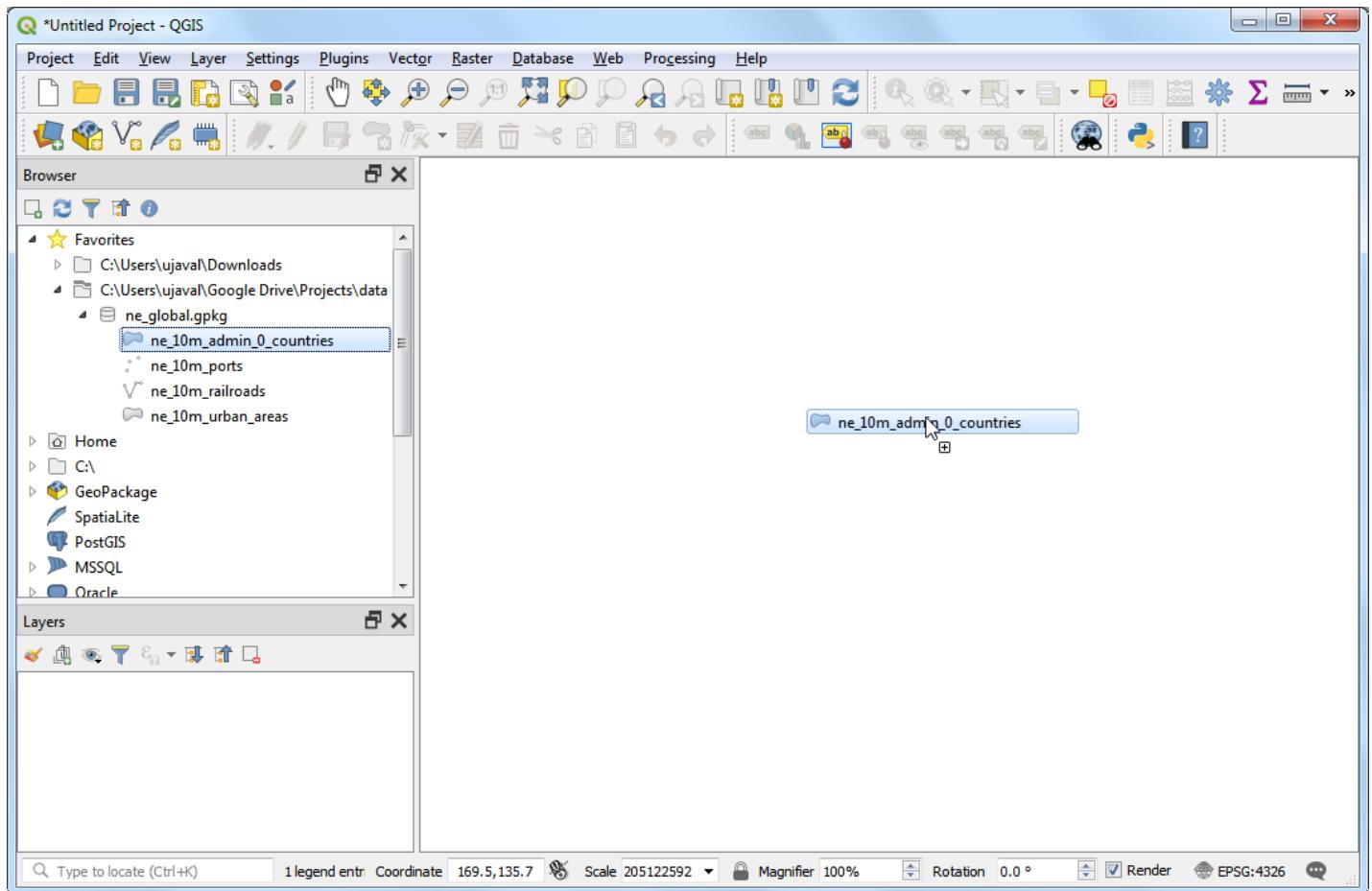
Data Source [NATURALEARTH]

For convenience, you may directly download a geopackage containing the above layer from below:

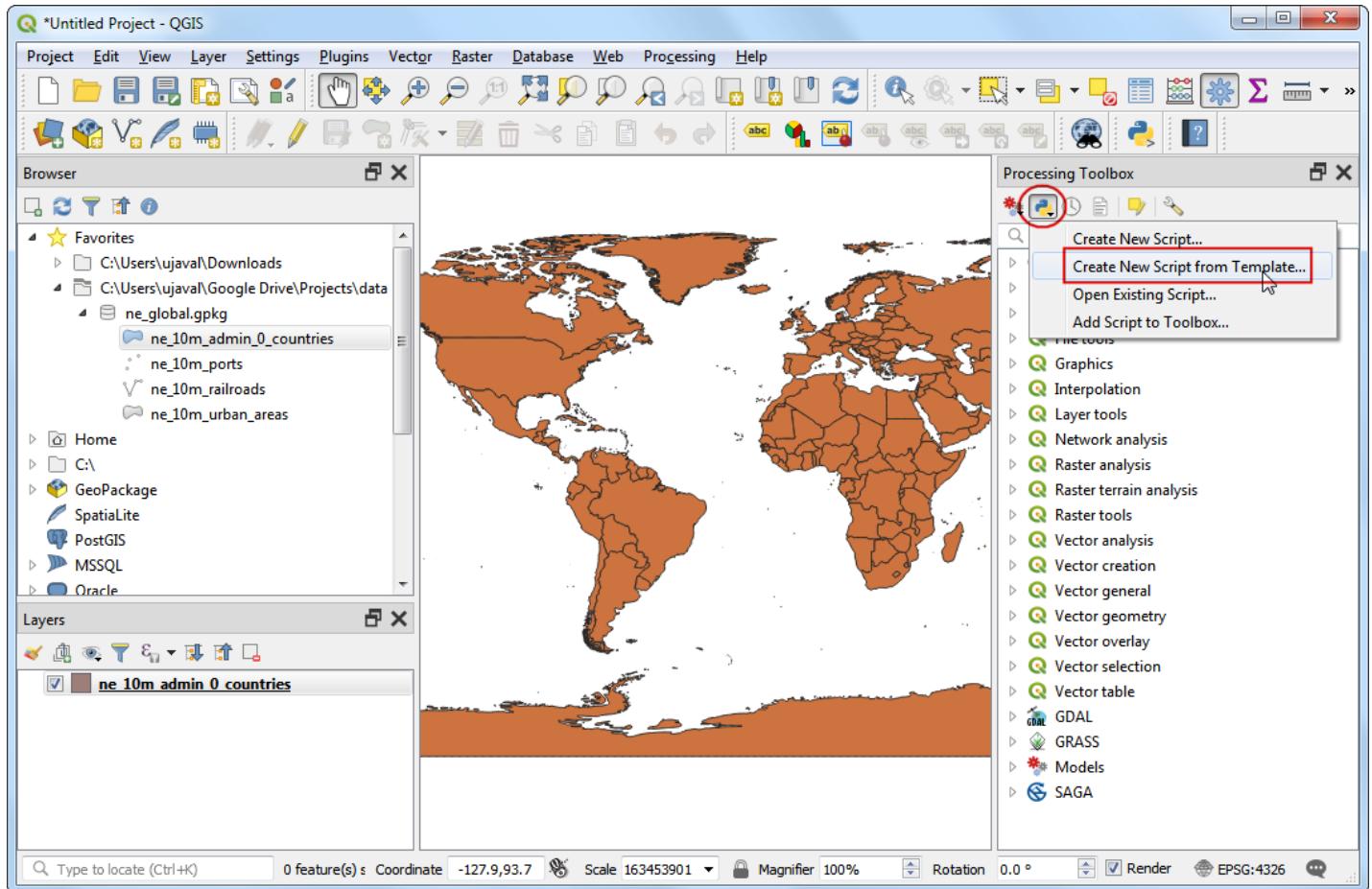
[ne_global.gpkg](#)

Procedure

1. In the QGIS Browser Panel, locate the directory where you saved your downloaded data. Expand the `.zip` or the `.gPKG` entry and select the `ne_10m_admin_0_countries` layer. Drag the layer to the canvas.



2. Go to Processing ▶ Toolbox. Click the Scripts button in the toolbar and select Create New Script from Template.

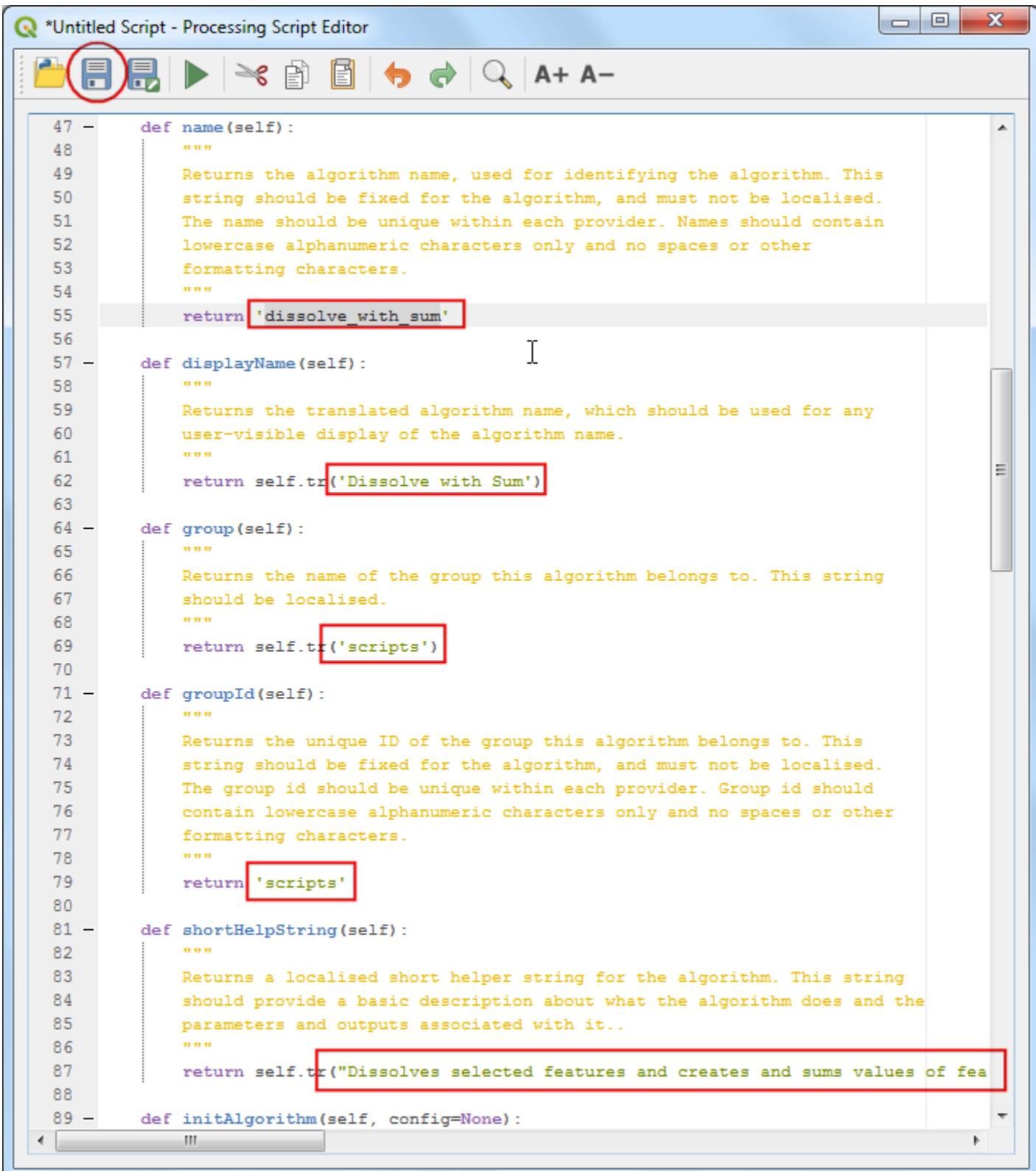


3. The template contains all the boilerplate code that is required for the Processing Framework to recognize it as a Processing script, and manage inputs/outputs. Let's start customizing the example template to our needs. First change the class name from `ExampleProcessingAlgorithm` to `DissolveProcessingAlgorithm`. This name also needs to be updated in the `createInstance` method. Add a docstring to the class that explains what the algorithm does.

The screenshot shows a Processing Script Editor window with the file `*dissolve_with_sum.py` open. The code defines a class `DissolveProcessingAlgorithm` that inherits from `QgsProcessingAlgorithm`. The class has methods `tr`, `createInstance`, and `name`. The `createInstance` method is highlighted with a red box. The `name` method is also highlighted with a red box.

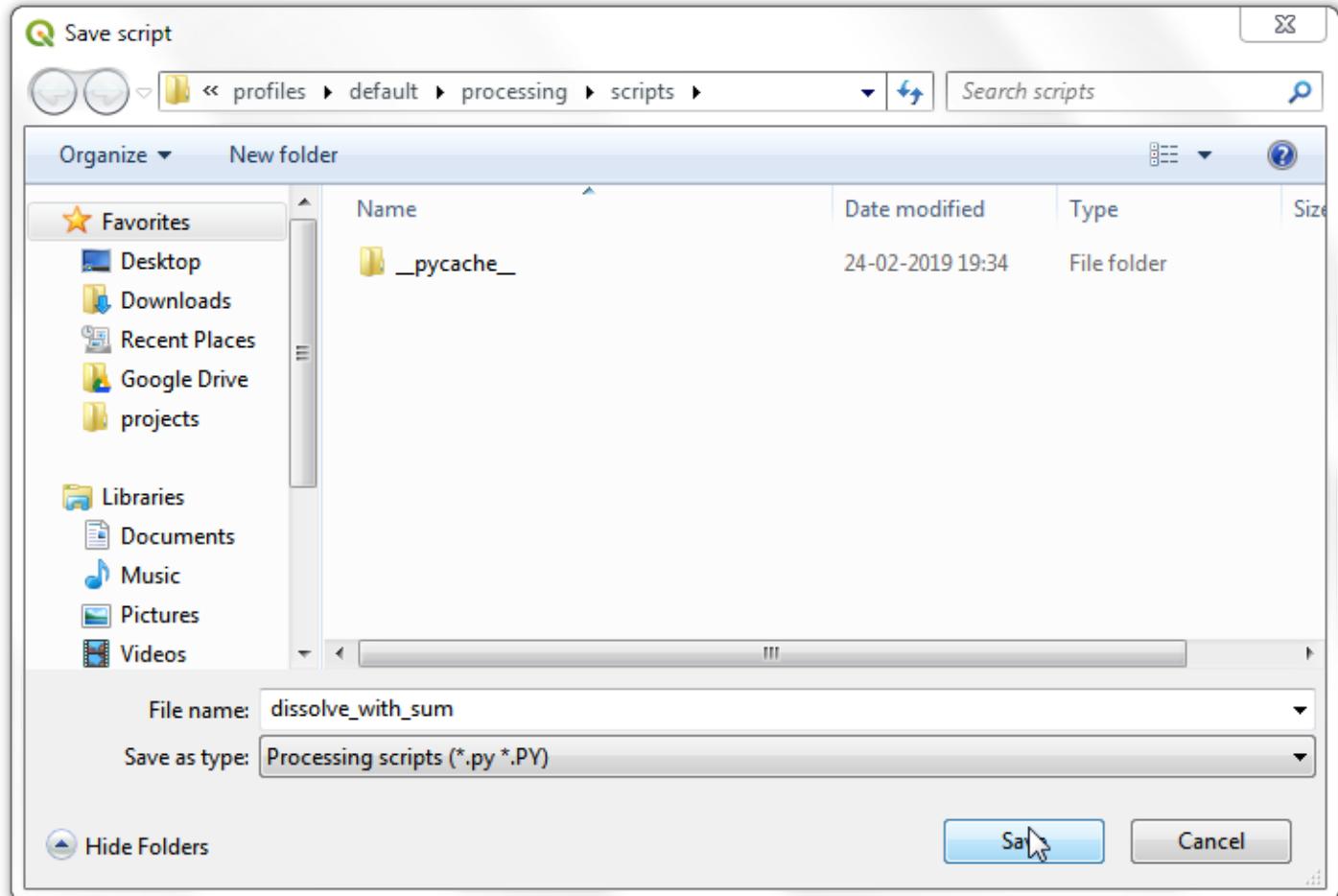
```
22     QgsProcessingParameterFeatureSink,
23 )
24 import processing
25
26
27 - class DissolveProcessingAlgorithm(QgsProcessingAlgorithm):
28     """
29     Dissolve algorithm that dissolves features based on selected
30     attribute and summarizes the selected field by computing the
31     sum of dissolved features.
32     """
33     INPUT = 'INPUT'
34     OUTPUT = 'OUTPUT'
35
36 -     def tr(self, string):
37         """
38             Returns a translatable string with the self.tr() function.
39         """
40         return QCoreApplication.translate('Processing', string)
41
42 -     def createInstance(self):
43         return DissolveProcessingAlgorithm()
44
45 -     def name(self):
46         """
47             Returns the algorithm name, used for identifying the algorithm. This
48             string should be fixed for the algorithm, and must not be localised.
49             The name should be unique within each provider. Names should contain
```

4. As you scroll down, you will see methods that assign name, group, description etc. to the script. Change the return values for `name` method to be `dissolve_with_sum`, `displayName` method to `Dissolve with Sum`, `group` method and `groupId` method to `scripts`. Change the return value of `shortHelpString` method to a description that will appear to the user. Click the Save button.



```
47 -     def name(self):
48 -         """
49 -             Returns the algorithm name, used for identifying the algorithm. This
50 -             string should be fixed for the algorithm, and must not be localised.
51 -             The name should be unique within each provider. Names should contain
52 -             lowercase alphanumeric characters only and no spaces or other
53 -             formatting characters.
54 -         """
55 -         return 'dissolve_with_sum'
56 -
57 -     def displayName(self):
58 -         """
59 -             Returns the translated algorithm name, which should be used for any
60 -             user-visible display of the algorithm name.
61 -         """
62 -         return self.tr('Dissolve with Sum')
63 -
64 -     def group(self):
65 -         """
66 -             Returns the name of the group this algorithm belongs to. This string
67 -             should be localised.
68 -         """
69 -         return self.tr('scripts')
70 -
71 -     def groupId(self):
72 -         """
73 -             Returns the unique ID of the group this algorithm belongs to. This
74 -             string should be fixed for the algorithm, and must not be localised.
75 -             The group id should be unique within each provider. Group id should
76 -             contain lowercase alphanumeric characters only and no spaces or other
77 -             formatting characters.
78 -         """
79 -         return 'scripts'
80 -
81 -     def shortHelpString(self):
82 -         """
83 -             Returns a localised short helper string for the algorithm. This string
84 -             should provide a basic description about what the algorithm does and the
85 -             parameters and outputs associated with it..
86 -         """
87 -         return self.tr("Dissolves selected features and creates and sums values of fea
88 -
89 -     def initAlgorithm(self, config=None):
90 -         !!!
```

5. Name the script `dissolve_with_sum` and save it at the default location under `profiles > default > processing > scripts` folder.



6. Now we will define the inputs for the script. The template already contains a definition of an INPUT Vector Layer and a OUTPUT layer. We will add 2 new inputs which allows the user to select a DISSOLVE_FIELD and a SUM_FIELD. Add a new import at the top and the following code in the `initAlgorithm` method. Click the Run button to preview the changes.

```
from qgis.core import QgsProcessingParameterField

self.addParameter(
    QgsProcessingParameterField(
        self.DISSOLVE_FIELD,
        'Choose Dissolve Field',
        '',
        self.INPUT))

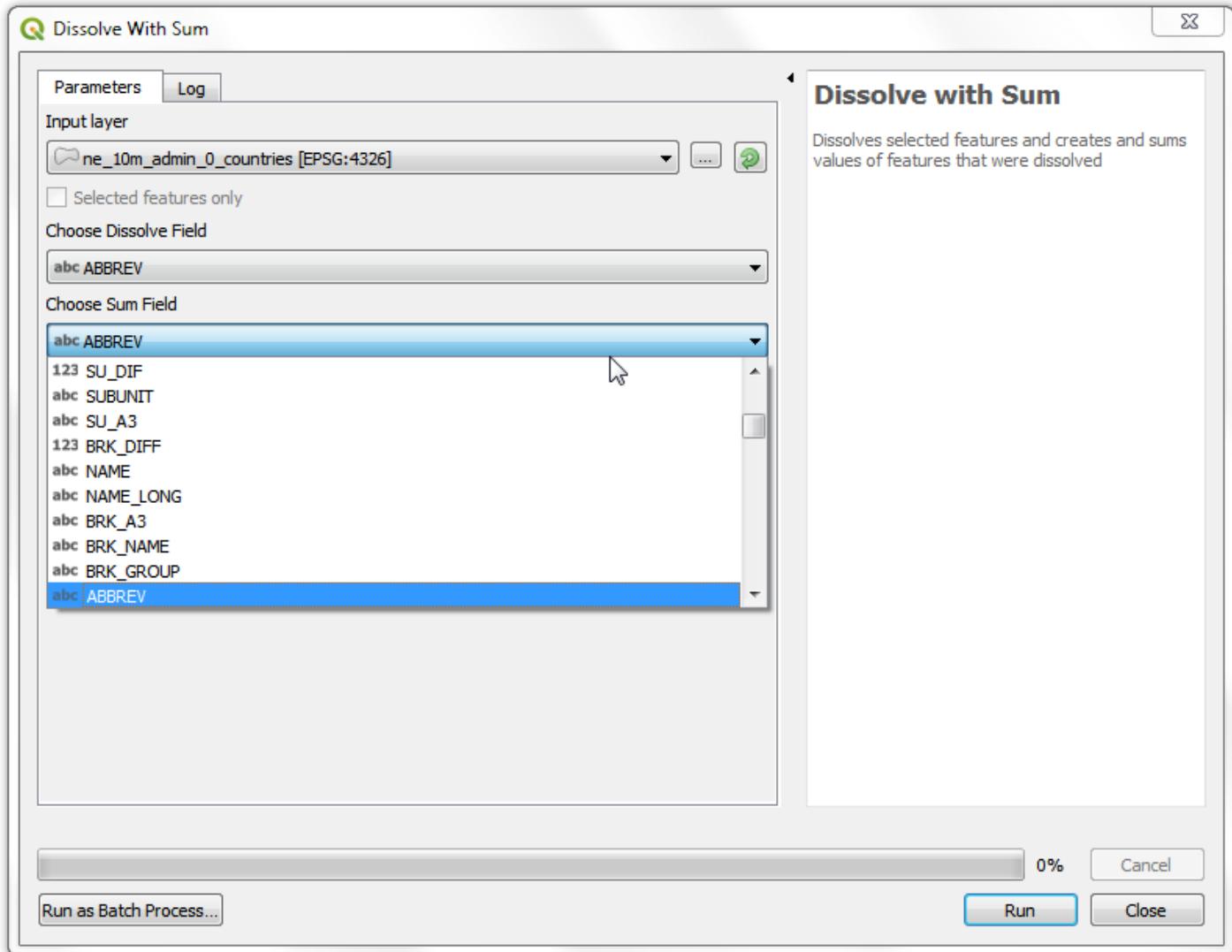
self.addParameter(
    QgsProcessingParameterField(
        self.SUM_FIELD,
        'Choose Sum Field',
        '',
        self.INPUT))
```

The screenshot shows the Processing Script Editor window with the file "dissolve_with_sum.py" open. The editor has a toolbar at the top with various icons for file operations like save, open, and run. A red circle highlights the play/run button icon. Below the toolbar is a code editor area with syntax highlighting for Python. The code defines a class `DissolveProcessingAlgorithm` that inherits from `QgsProcessingAlgorithm`. It imports several modules and defines constants for input, output, and fields. Two specific lines of code are highlighted with red boxes: the import of `QgsProcessingParameterField` and the definition of the `DISSOLVE_FIELD` constant.

```
13
14     from PyQt5.QtCore import QCoreApplication
15 -    from qgis.core import (QgsProcessing,
16 -                            QgsFeatureSink,
17 -                            QgsProcessingException,
18 -                            QgsProcessingAlgorithm,
19 -                            QgsProcessingParameterFeatureSource,
20 -                            QgsProcessingParameterFeatureSink,
21 -                            QgsProcessingParameterField)
22     import processing
23
24
25 - class DissolveProcessingAlgorithm(QgsProcessingAlgorithm):
26 -     """
27 -         Dissolve algorithm that dissolves features based on selected
28 -         attribute and summarizes the selected field by computing the
29 -         sum of dissolved features.
30 -     """
31     INPUT = 'INPUT'
32     OUTPUT = 'OUTPUT'
33     DISSOLVE_FIELD = 'dissolve_field'
34     SUM_FIELD = 'sum_field'
35
36 -     def tr(self, string):
37 -         return QCoreApplication.translate('Processing', string)
```

```
85     return self.tr("Dissolves selected features and creates and sums values of features")
86
87     def initAlgorithm(self, config=None):
88         """
89             Here we define the inputs and output of the algorithm, along
90             with some other properties.
91         """
92
93         # We add the input vector features source. It can have any kind of
94         # geometry.
95         self.addParameter(
96             QgsProcessingParameterFeatureSource(
97                 self.INPUT,
98                 self.tr('Input layer'),
99                 [QgsProcessing.TypeVectorAnyGeometry]
100            )
101        )
102
103        self.addParameter(
104            QgsProcessingParameterField(
105                self.DISSOLVE_FIELD,
106                'Choose Dissolve Field',
107                '',
108                self.INPUT))
109        self.addParameter(
110            QgsProcessingParameterField(
111                self.SUM_FIELD,
112                'Choose Sum Field',
113                '',
114                self.INPUT))
```

7. You will see a Dissolve with Sum dialog with our newly defined inputs. Select the `ne_10m_admin_0_countries` layer as Input layer. As both Dissolve Field and Sum Fields are filtered based on the input layer, they will be pre-populated with existing fields from the input layer. Click the Close button.



8. Now we define our custom logic for processing data in the `processAlgorithm` method. This method gets passed a dictionary called `parameters`. It contains the inputs that the user has selected. There are helper methods that allow you to take these inputs and create appropriate objects. We first get our inputs using `parameterAsSource` and `parameterAsString` methods. Next we want to create a feature sink where we will write the output. QGIS3 has a new class called `QgsFeatureSink` which is the preferred way to create objects that can accept new features. The output needs only 2 fields - one for the value of dissolved field and another for the sum of the selected field.

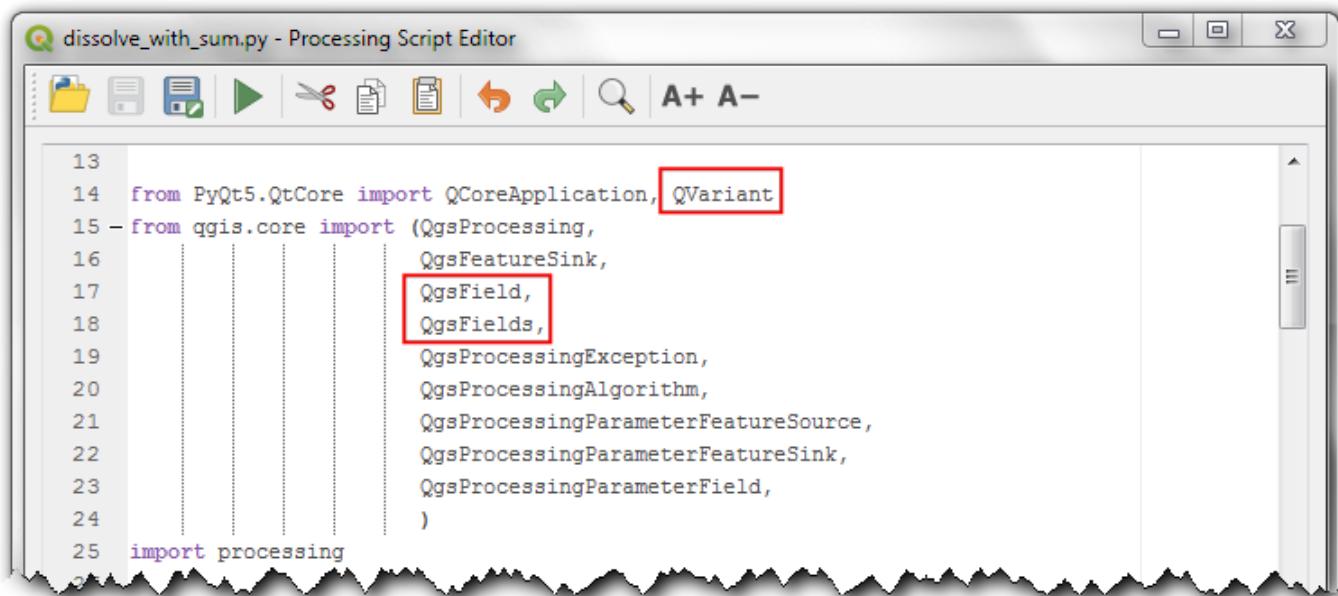
```
from PyQt5.QtCore import QVariant
from qgis.core import QgsField, QgsFields

source = self.parameterAsSource(
    parameters,
    self.INPUT,
    context)
dissolve_field = self.parameterAsString(
    parameters,
    self.DISSOLVE_FIELD,
```

```
    context)
sum_field = self.parameterAsString(
    parameters,
    self.SUM_FIELD,
    context)

fields = QgsFields()
fields.append(QgsField(dissolve_field, QVariant.String))
fields.append(QgsField('SUM_' + sum_field, QVariant.Double))

(sink, dest_id) = self.parameterAsSink(
    parameters,
    self.OUTPUT,
    context, fields, source.wkbType(), source.sourceCrs())
```



```

125
126     def processAlgorithm(self, parameters, context, feedback):
127         """
128             Here is where the processing itself takes place.
129         """
130         source = self.parameterAsSource(
131             parameters,
132             self.INPUT,
133             context
134         )
135         dissolve_field = self.parameterAsString(
136             parameters,
137             self.DISSOLVE_FIELD,
138             context)
139         sum_field = self.parameterAsString(
140             parameters,
141             self.SUM_FIELD,
142             context)
143
144         fields = QgsFields()
145         fields.append(QgsField(dissolve_field, QVariant.String))
146         fields.append(QgsField('SUM_' + sum_field, QVariant.Double))
147
148         (sink, dest_id) = self.parameterAsSink(
149             parameters,
150             self.OUTPUT,
151             context, fields, source.wkbType(), source.sourceCrs())

```

9. Now we will ready the input features and create a dictionary to hold the unique values from the dissolve_field and the sum of the values from the sum_field. Note the use of `feedback.pushInfo()` method to communicate the status with the user.

```

feedback.pushInfo('Extracting unique values from dissolve_field and computing sum')

features = source.getFeatures()
unique_values = set(f[dissolve_field] for f in features)

# Get Indices of dissolve field and sum field
dissolveIdx = source.fields().indexFromName(dissolve_field)
sumIdx = source.fields().indexFromName(sum_field)

# Find all unique values for the given dissolve_field and
# sum the corresponding values from the sum_field
sum_unique_values = {}
attrs = [{dissolve_field: f[dissolveIdx], sum_field: f[sumIdx]} for f in source.getFeatures()]
for unique_value in unique_values:
    val_list = [f_attr[sum_field] for f_attr in attrs if f_attr[dissolve_field] == unique_value]
    sum_unique_values[unique_value] = sum(val_list)

```

```
145     fields.append(QgsField(dissolve_field, QVariant.String))
146     fields.append(QgsField('SUM_' + sum_field, QVariant.Double))
147
148 -     (sink, dest_id) = self.parameterAsSink(
149         parameters,
150         self.OUTPUT,
151         context, fields, source.wkbType(), source.sourceCrs())
152
153 # Create a dictionary to hold the unique values from the
154 # dissolve_field and the sum of the values from the sum_field
155 feedback.pushInfo('Extracting unique values from dissolve_field and computing sum')
156 features = source.getFeatures()
157 unique_values = set(f[dissolve_field] for f in features)
158 # Get Indices of dissolve field and sum field
159 dissolveIdx = source.fields().indexFromName(dissolve_field)
160 sumIdx = source.fields().indexFromName(sum_field)
161
162 # Find all unique values for the given dissolve_field and
163 # sum the corresponding values from the sum_field
164 sum_unique_values = {}
165 attrs = [{dissolve_field: f[dissolveIdx], sum_field: f[sumIdx]}
166           for f in source.getFeatures()]
167 for unique_value in unique_values:
168     val_list = [f_attr[sum_field]
169                 for f_attr in attrs if f_attr[dissolve_field] == unique_value]
170     sum_unique_values[unique_value] = sum(val_list)
171
172 return {self.OUTPUT: dest_id}
```

10. Next, we will call the built-in processing algorithm `native:dissolve` on the input layer to generate the dissolved geometries. Once we have the dissolved geometries, we iterate through the output of the dissolve algorithm and create new features to be added to the output. At the end we return the `dest_id` FeatureSink as the output. Now the script is ready. Click the Run button.

Note

Notice the use of `parameters[self.INPUT]` to fetch the input layer from the parameters dictionary directly without defining it as a source. As we are passing the input object to the algorithm without doing anything with it, it's not necessary to define it as a source.

```
from qgis.core import QgsFeature
```

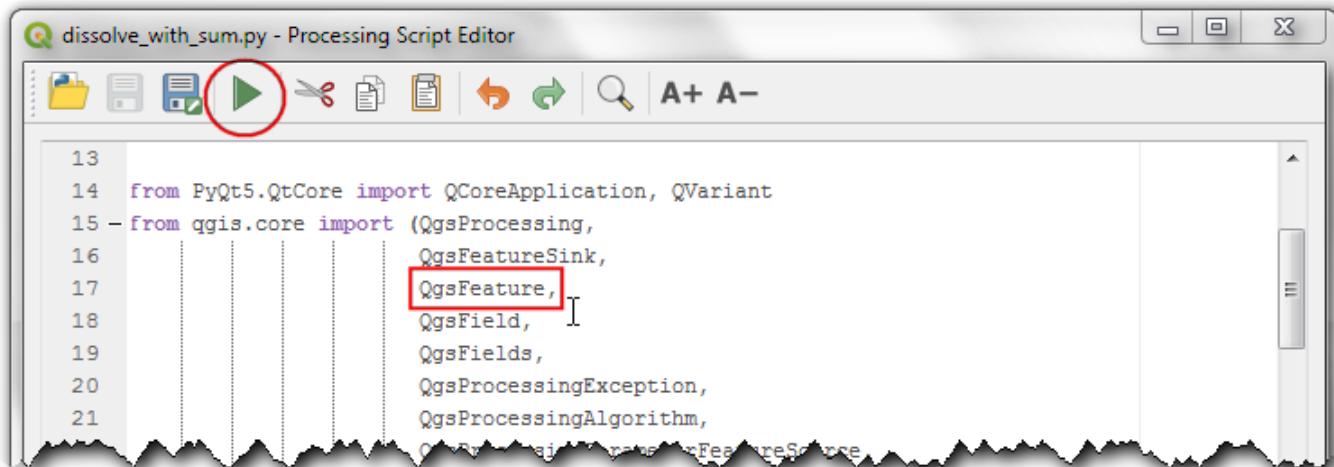
```

# Running the processing dissolve algorithm
feedback.pushInfo('Dissolving features')
dissolved_layer = processing.run("native:dissolve", {
    'INPUT': parameters[self.INPUT],
    'FIELD': dissolve_field,
    'OUTPUT': 'memory:'
}, context=context, feedback=feedback)['OUTPUT']

# Read the dissolved layer and create output features
for f in dissolved_layer.getFeatures():
    new_feature = QgsFeature()
    # Set geometry to dissolved geometry
    new_feature.setGeometry(f.geometry())
    # Set attributes from sum_unique_values dictionary that we had computed
    new_feature.setAttributes([f[dissolve_field], sum_unique_values[f[dissolve_field]]])
    sink.addFeature(new_feature, QgsFeatureSink.FastInsert)

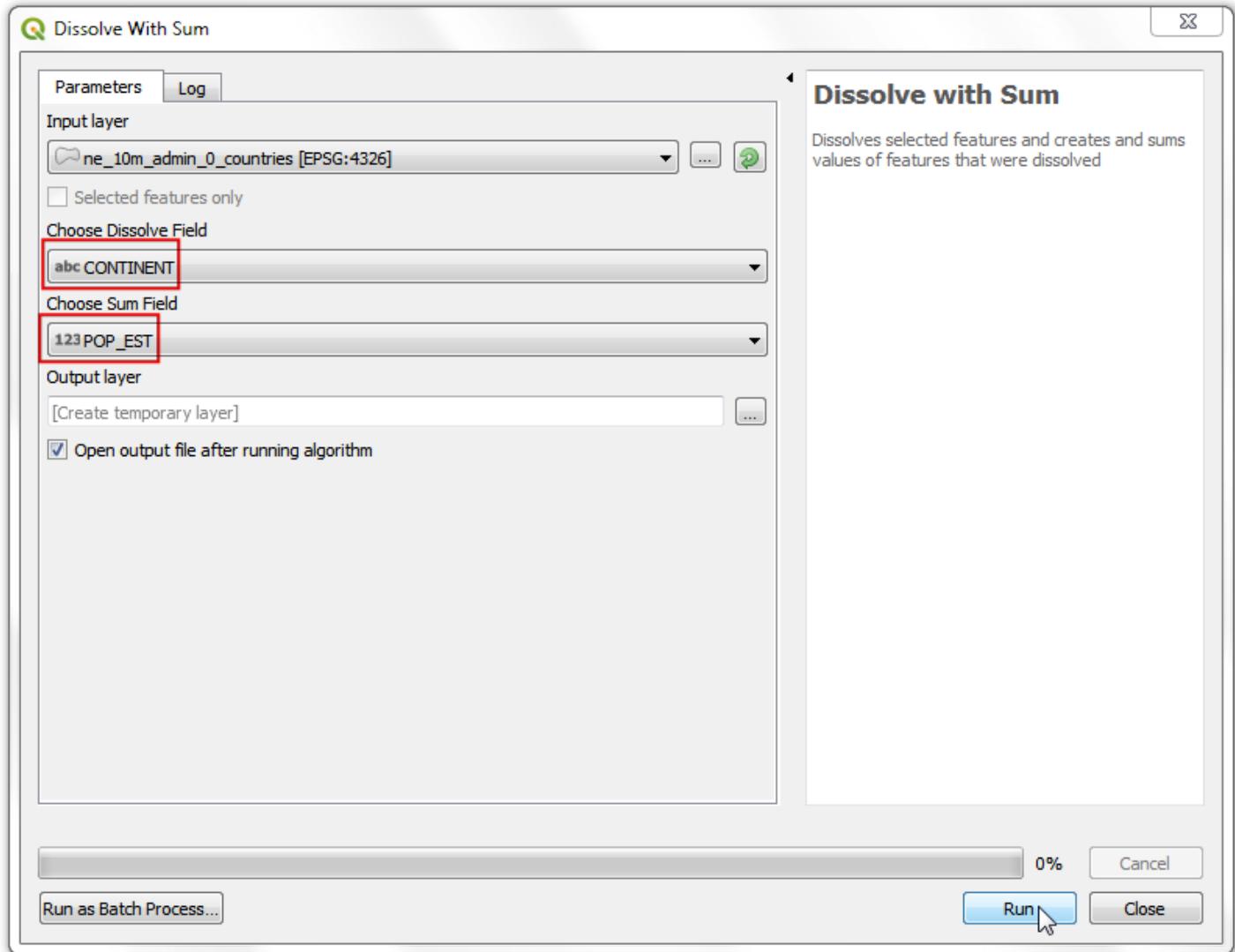
return {self.OUTPUT: dest_id}

```

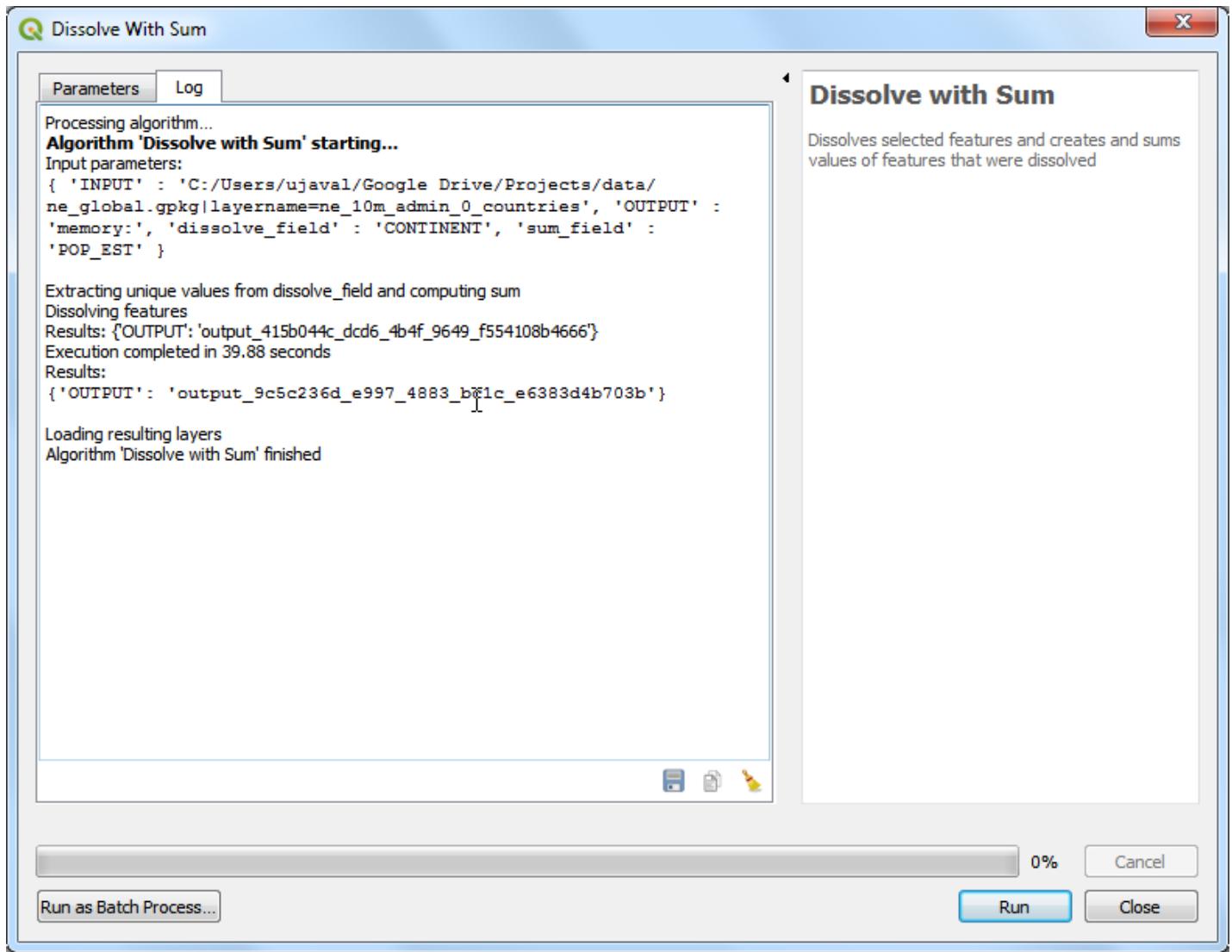


```
169         for v in attrs if v_attr[dissolve_field] != unique_value]
170             sum_unique_values[unique_value] = sum(val_list)
171
172     # Running the processing dissolve algorithm
173     feedback.pushInfo('Dissolving features')
174     dissolved_layer = processing.run("native:dissolve", {
175         'INPUT': parameters[self.INPUT],
176         'FIELD': dissolve_field,
177         'OUTPUT': 'memory:'
178     }, context=context, feedback=feedback)['OUTPUT']
179
180     # Read the dissolved layer and create output features
181     for f in dissolved_layer.getFeatures():
182         new_feature = QgsFeature()
183         # Set geometry to dissolved geometry
184         new_feature.setGeometry(f.geometry())
185         # Set attributes from sum_unique_values dictionary that we had computed
186         new_feature.setAttributes([f[dissolve_field], sum_unique_values[f[dissolve_field]]])
187         sink.addFeature(new_feature, QgsFeatureSink.FastInsert)
188
189     return {self.OUTPUT: dest_id}
```

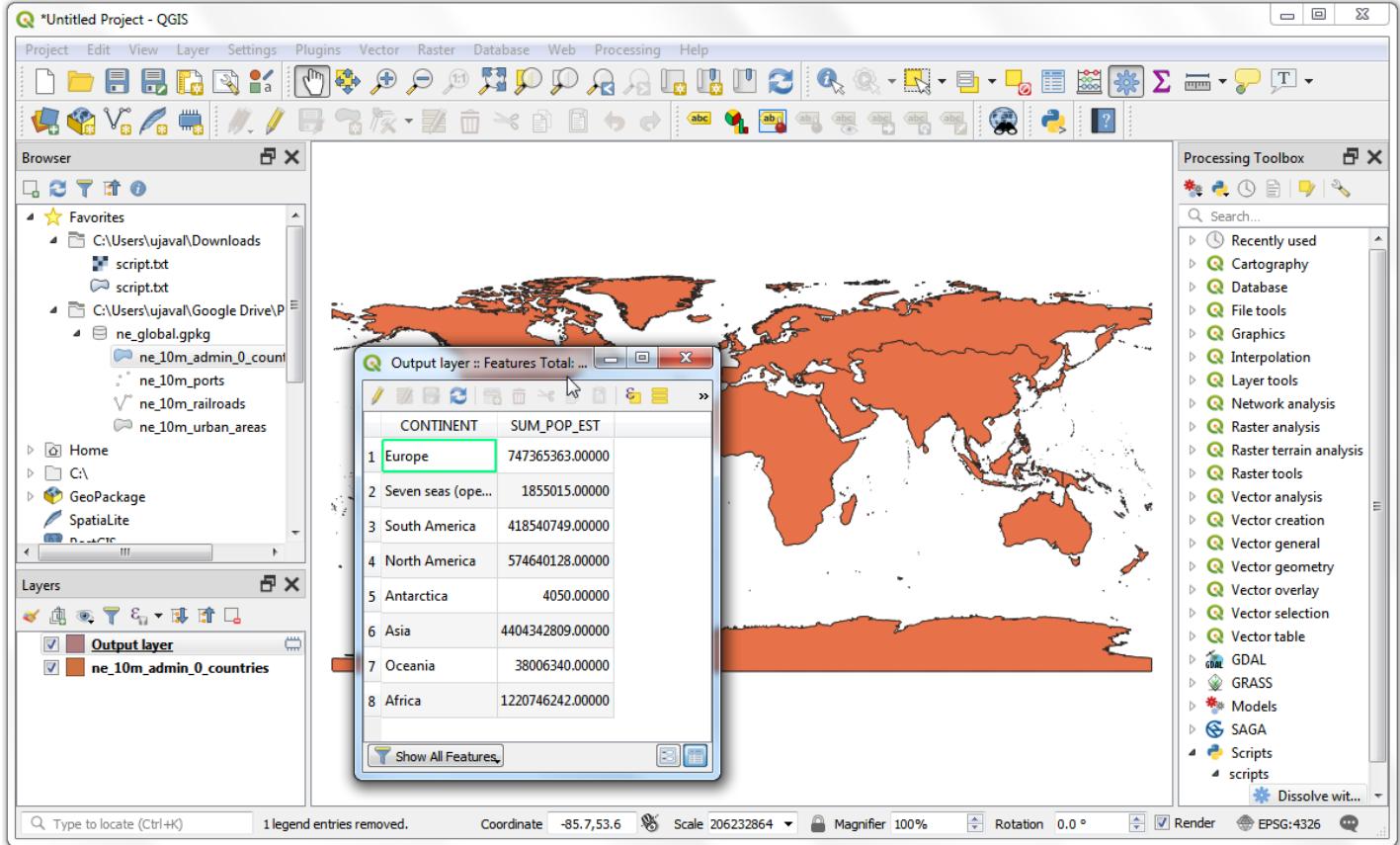
11. In the Dissolve with Sum, dialog, select ne_10m_admin_0_countries as the Input layer, CONTINENT as the Dissolve field and POP_EST as the Sum field. Click Run.



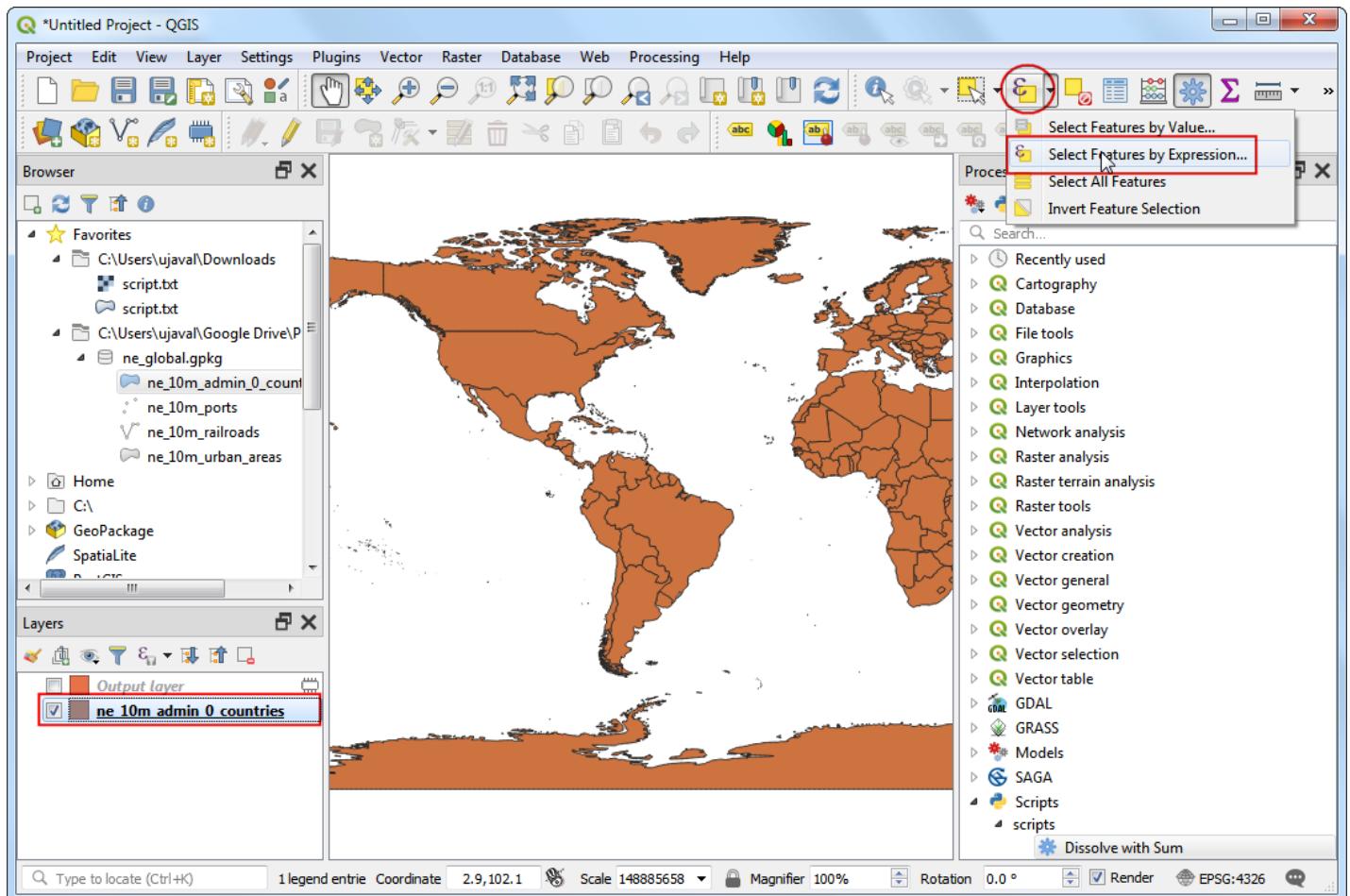
12. Once the processing is finished, click the Close button and switch to the main QGIS window.



13. You will see the dissolved output layer with one feature for every continent and the total population summed from the individual countries belonging to that continent.

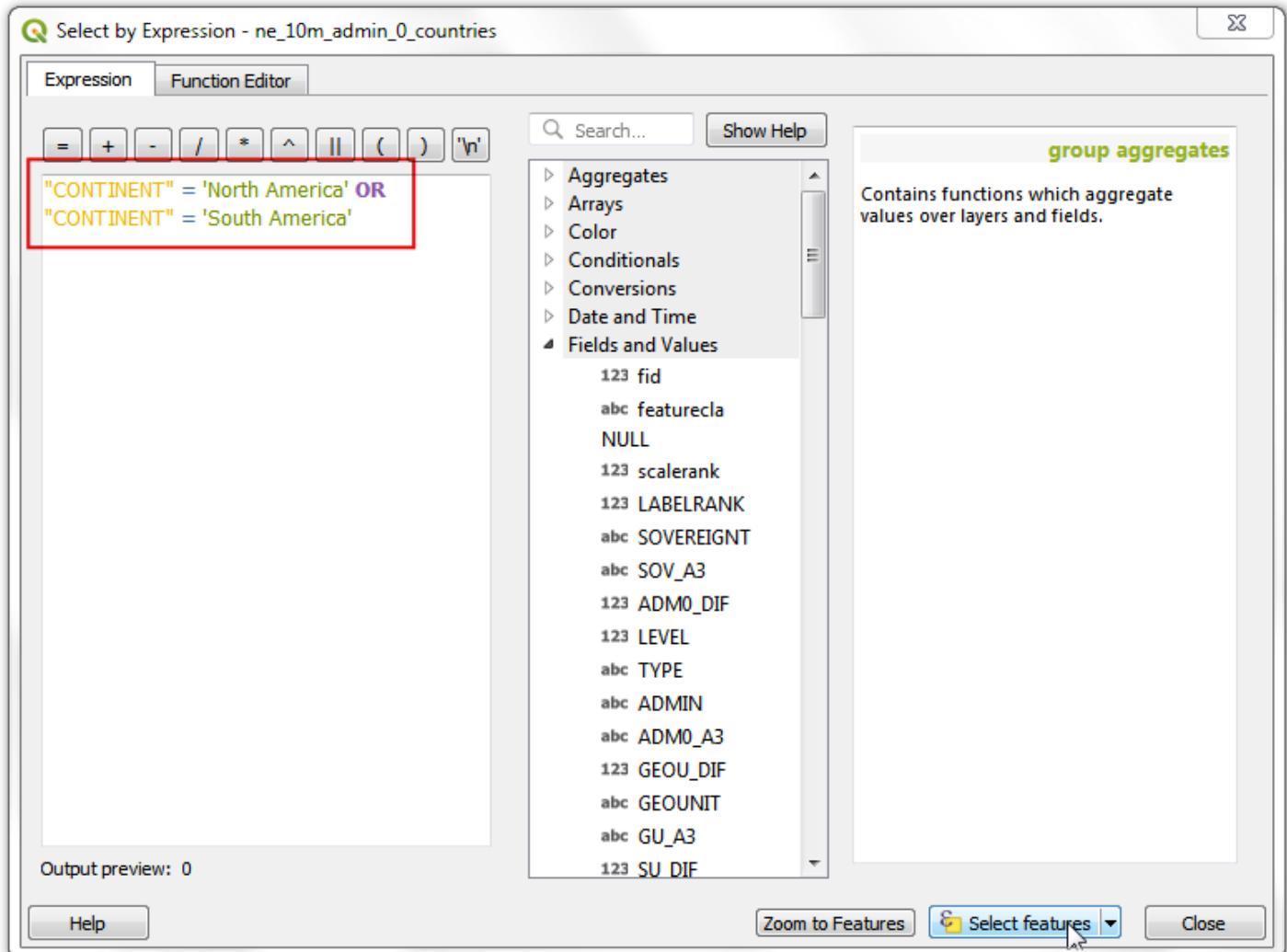


14. One another advantage of writing processing script is that the methods within the Processing Framework are aware of layer selection and automatically filter your inputs to use only the selected features. This happens because we are defining our input as a `QgsProcessingParameterFeatureSource`. A feature source allows use of ANY object which contains vector features, not just a vector layer, so when there are selected features in your layer and ask Processing to use selected features, the input is passed on to your script as a `QgsProcessingFeatureSource` object containing selected features and not the full vector layer. Here's a quick demonstration of this functionality. Let's say we want to dissolve only certain continents. Let's create a selection using Select feature by Expression tool.

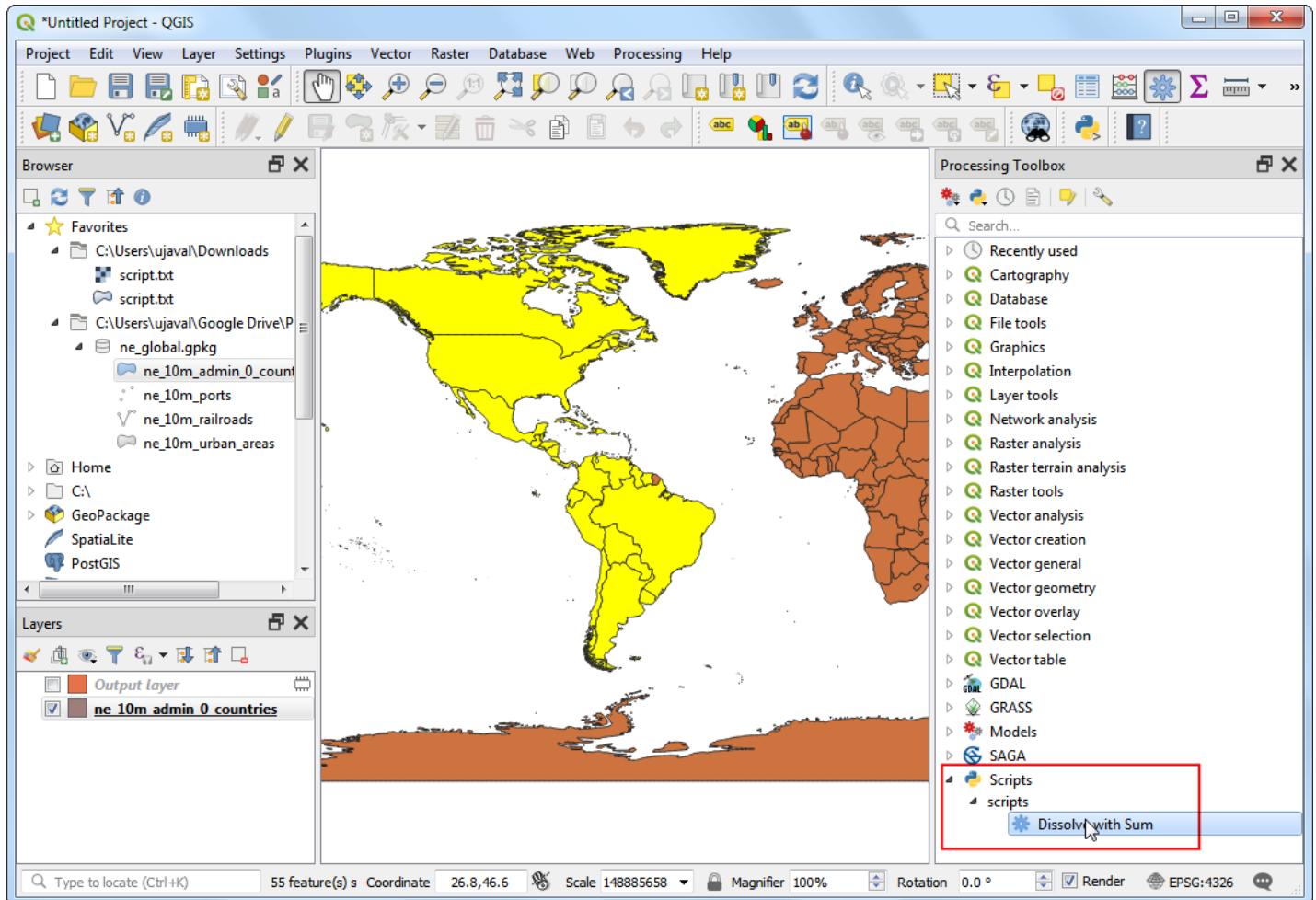


15. Enter the following expression to select features from North and South America and click Select.

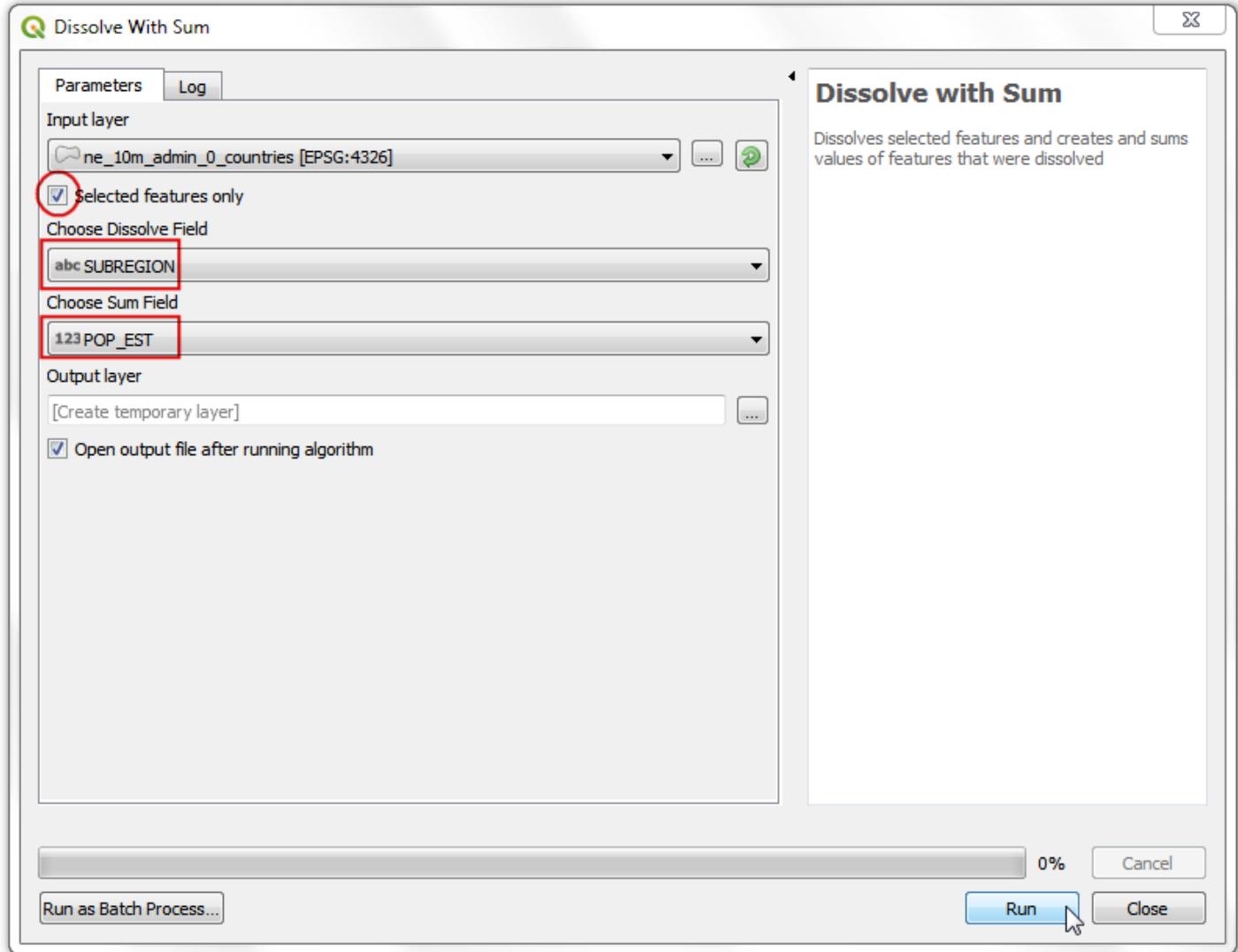
```
"CONTINENT" = 'North America' OR "CONTINENT" = 'South America'
```



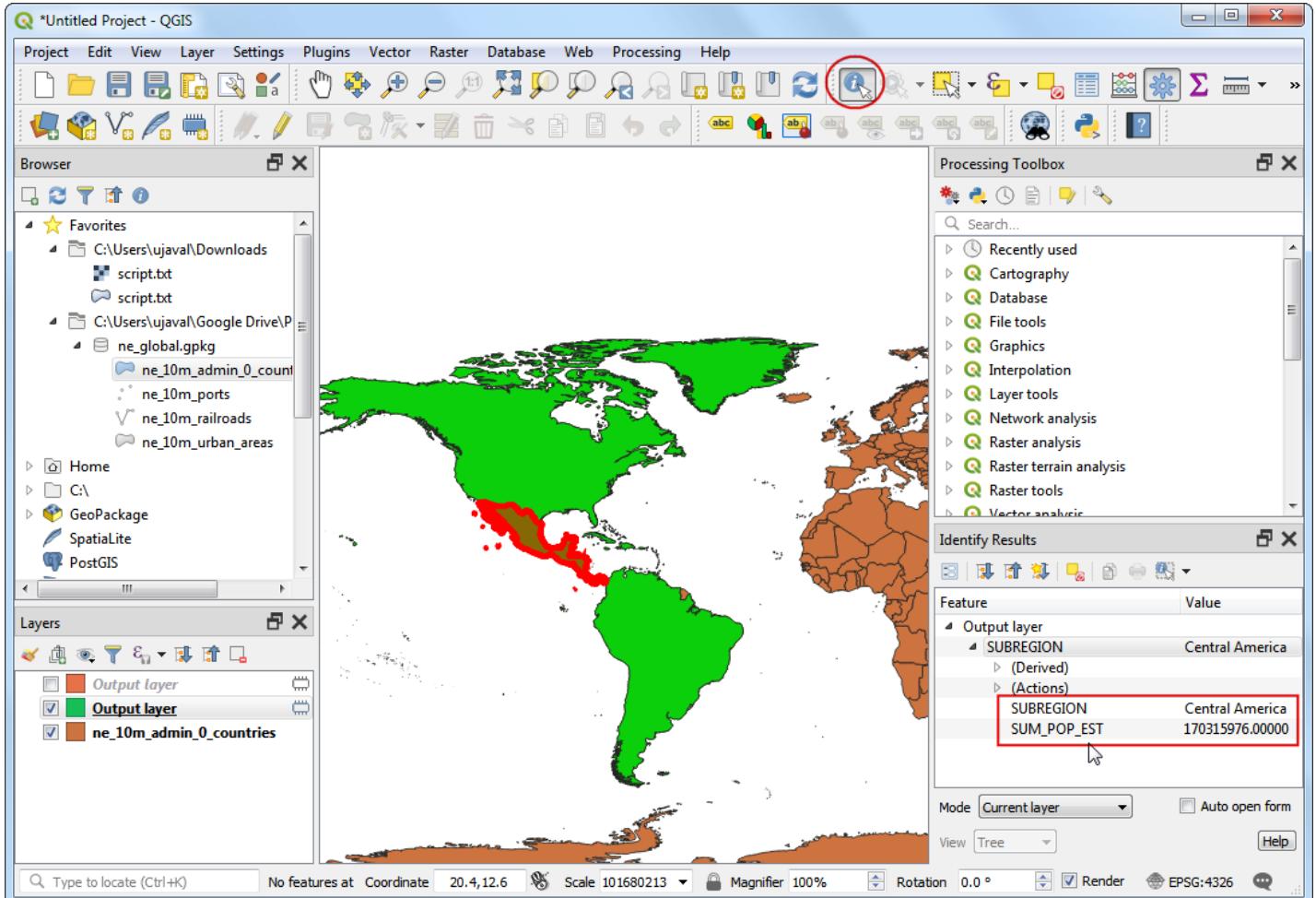
16. You will see the selected features highlighted in yellow. Locate the dissolve_with_sum script and double-click it to run it.



17. In the Dissolve with Sum dialog, select the `ne_10m_admin_0_countries` as the Input layer. This time, make sure you check the Selected features only box. Choose `SUBREGION` as the Dissolve field and `POP_EST` as the Sum field.



18. Once the processing finishes, click Close and switch back to the main QGIS window. You will notice a new layer with only the selected features dissolved. Click Identify button and click on a feature to inspect and verify that the script worked correctly.



Below is the complete script for reference. You may modify it to suit your needs.

```

        QgsProcessingParameterField,
    )

import processing

class DissolveProcessingAlgorithm(QgsProcessingAlgorithm):
    """
    Dissolve algorithm that dissolves features based on selected
    attribute and summarizes the selected field by computing the
    sum of dissolved features.
    """
    INPUT = 'INPUT'
    OUTPUT = 'OUTPUT'
    DISSOLVE_FIELD = 'dissolve_field'
    SUM_FIELD = 'sum_field'

    def tr(self, string):
        """
        Returns a translatable string with the self.tr() function.
        """
        return QCoreApplication.translate('Processing', string)

    def createInstance(self):
        return DissolveProcessingAlgorithm()

    def name(self):
        """
        Returns the algorithm name, used for identifying the algorithm. This
        string should be fixed for the algorithm, and must not be localised.
        The name should be unique within each provider. Names should contain
        lowercase alphanumeric characters only and no spaces or other
        formatting characters.
        """
        return 'dissolve_with_sum'

    def displayName(self):
        """
        Returns the translated algorithm name, which should be used for any
        user-visible display of the algorithm name.
        """
        return self.tr('Dissolve with Sum')

    def group(self):
        """
        Returns the name of the group this algorithm belongs to. This string
        should be localised.
        """
        return self.tr('scripts')

    def groupId(self):
        """
        Returns the unique ID of the group this algorithm belongs to. This
        string should be fixed for the algorithm, and must not be localised.
        The group id should be unique within each provider. Group id should
        contain lowercase alphanumeric characters only and no spaces or other
        formatting characters.
        """

```

```

    return 'scripts'

def shortHelpString(self):
    """
    Returns a localised short helper string for the algorithm. This string
    should provide a basic description about what the algorithm does and the
    parameters and outputs associated with it..
    """
    return self.tr("Dissolves selected features and creates and sums values of features")

def initAlgorithm(self, config=None):
    """
    Here we define the inputs and output of the algorithm, along
    with some other properties.
    """
    # We add the input vector features source. It can have any kind of
    # geometry.
    self.addParameter(
        QgsProcessingParameterFeatureSource(
            self.INPUT,
            self.tr('Input layer'),
            [QgsProcessing.TypeVectorAnyGeometry]
        )
    )
    self.addParameter(
        QgsProcessingParameterField(
            self.DISSOLVE_FIELD,
            'Choose Dissolve Field',
            '',
            self.INPUT))
    self.addParameter(
        QgsProcessingParameterField(
            self.SUM_FIELD,
            'Choose Sum Field',
            '',
            self.INPUT))
    # We add a feature sink in which to store our processed features (this
    # usually takes the form of a newly created vector layer when the
    # algorithm is run in QGIS).
    self.addParameter(
        QgsProcessingParameterFeatureSink(
            self.OUTPUT,
            self.tr('Output layer'))
    )

def processAlgorithm(self, parameters, context, feedback):
    """
    Here is where the processing itself takes place.
    """
    source = self.parameterAsSource(
        parameters,
        self.INPUT,
        context
    )
    dissolve_field = self.parameterAsString(
        parameters,

```

```

        self.DISSOLVE_FIELD,
        context)
sum_field = self.parameterAsString(
    parameters,
    self.SUM_FIELD,
    context)

fields = QgsFields()
fields.append(QgsField(dissolve_field, QVariant.String))
fields.append(QgsField('SUM_' + sum_field, QVariant.Double))

(sink, dest_id) = self.parameterAsSink(
    parameters,
    self.OUTPUT,
    context, fields, source.wkbType(), source.sourceCrs())

# Create a dictionary to hold the unique values from the
# dissolve_field and the sum of the values from the sum_field
feedback.pushInfo('Extracting unique values from dissolve_field and computing sum')
features = source.getFeatures()
unique_values = set(f[dissolve_field] for f in features)
# Get Indices of dissolve field and sum field
dissolveIdx = source.fields().indexFromName(dissolve_field)
sumIdx = source.fields().indexFromName(sum_field)

# Find all unique values for the given dissolve_field and
# sum the corresponding values from the sum_field
sum_unique_values = {}
attrs = [{dissolve_field: f[dissolveIdx], sum_field: f[sumIdx]}
         for f in source.getFeatures()]
for unique_value in unique_values:
    val_list = [f_attr[sum_field]
                for f_attr in attrs if f_attr[dissolve_field] == unique_value]
    sum_unique_values[unique_value] = sum(val_list)

# Running the processing dissolve algorithm
feedback.pushInfo('Dissolving features')
dissolved_layer = processing.run("native:dissolve", {
    'INPUT': parameters[self.INPUT],
    'FIELD': dissolve_field,
    'OUTPUT': 'memory:'}
), context=context, feedback=feedback)['OUTPUT']

# Read the dissolved layer and create output features
for f in dissolved_layer.getFeatures():
    new_feature = QgsFeature()
    # Set geometry to dissolved geometry
    new_feature.setGeometry(f.geometry())
    # Set attributes from sum_unique_values dictionary that we had computed
    new_feature.setAttributes([f[dissolve_field], sum_unique_values[f[dissolve_field]]])
    sink.addFeature(new_feature, QgsFeatureSink.FastInsert)

return {self.OUTPUT: dest_id}

```