



Utrecht University

Applied Data Science Master's degree programme

Spatial Data Analysis and Simulation Modelling

Instruction manual for Laboratory 3.2:

Euclidean distance and Density

Document version: 0.1

Document modified: 2020.11.22

Dr. Simon Scheider, Haiqi Xu

Department of Human Geography and Spatial Planning

Faculty of Geosciences

s.scheider@uu.nl, h.xu1@uu.nl

Table of Contents

Euclidean distance and Density analysis	3
Set up QGIS analysis environment	4
Task 3 Distance-based analysis of park accessibility	5
Selection of parks	5
Proximity raster	8
Aggregate distance raster by zonal map algebra	11
Task 4 Density of parks in focal neighborhoods	12
Neighbor statistics	12
Zonal statistics	13
Alternative version of task 4: Computing the density of point data	15
Assignment: Analyze new datasource and document workflows of Lab 3.1 and 3.2	16
Optional Exercise: Python for GIS Analysis	17
Set up QGIS Python environment	17
Set up batch file	17
New python project	18
Generate a script with reusable functions for aggregation analysis	18
Program a function for areal interpolation from CBS data	20
Optional exercise: Python for other GIS analysis workflow	22

Euclidean distance and Density analysis

In Lab 3.2, you will continue to analyze the suitability of the city for older people in Amsterdam by using different GIS methods to summarize data into PC4 areas. This will allow you to assess the accessibility of Amsterdam. You still use QGIS Model Designer to implement all techniques as executable workflows.

Your task in this assignment is to compute the following characteristics on the level of postcode areas of level 4:

1. People would like to make sure they do not have to walk too long in order to reach a park for strolling. Which PC4 areas have the lowest average distances to parks?
2. Assess the average density of green or sport facilities in a 1km² window around each location inside every PC 4 area. This gives a more accurate picture of the density of greenness/facilities within an area.

The tasks additionally require some *distance and density based analysis* compared with Lab 3.1. Through these practicals, you will acquire the following *GIS competences*:

- Generate *workflows* in Model Designer and execute them
- Learn how to make use of Selections on a vector layer
- Learn *basic analytical workflows* that can be reused for many similar tasks (compare lecture):
 1. Aggregate distance to objects into spatial regions
 2. Aggregate density of objects into spatial regions


In particular, the following QGIS tools are used and practiced:

- General Vector tools
 - “Graduated Color” (Choropleth mapping)
 - “Select Layer by Attribute”
- Map algebra (raster) tools
 - “Rasterize”
 - “Zonal Statistics”
- Distance-based analysis
 - “Proximity (raster distance)”
- Density-based analysis
 - “r.neighbor”
 - “Heatmap”

Set up QGIS analysis environment

You will use the same dataset “dataAmsterdam” as Lab3.1.

To set up the QGIS analysis environment for Lab 3.2, you need to do the following settings:

- Open QGIS Desktop 3.14 and create an empty QGIS project
- Set Coordinate Reference System of the project as “Amersfoort / RD New” (EPSG:28992). (**Project** → **Properties...** → **CRS** tab)
- Click **Save** button  in the toolbar, create a new folder “Lab3.2 solution” and save the project as “Lab3.2” in the folder.

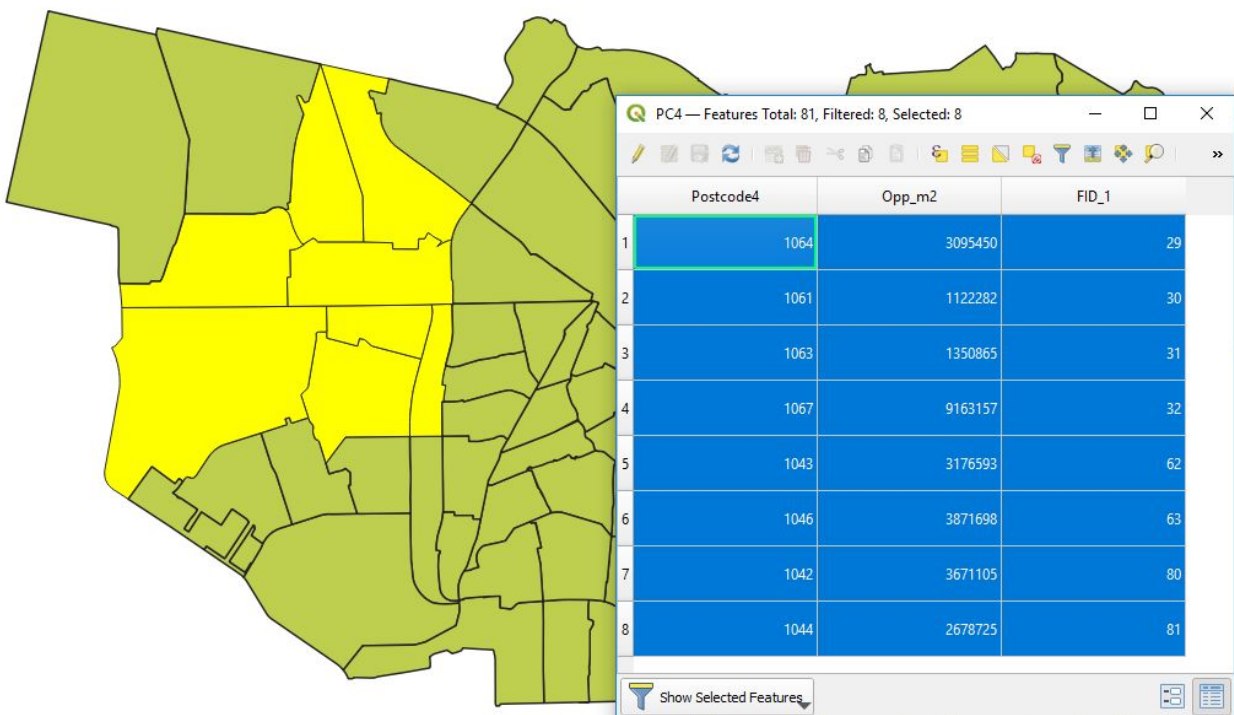
Task 3 Distance-based analysis of park accessibility

In this task, we are interested in assessing the accessibility of park areas for old people. From the “park area” map in Task 2, you can discover that there are several PC4 areas that are very unevenly covered by parks. For this reason, many inhabitants will have a hard time reaching these parks by foot. To assess this difficulty, we will compute the distance to the nearest park over the entire area and aggregate this information into PC4 areas.

For this purpose, we will have to select landuse areas in a similar way as in the Task 2, and then use distance-based analysis to generate a distance raster which needs to be aggregated into the PC4 areas. To make things more interesting, this time we will use a different (vector-based) landuse selection method. To start, drag “GRONDGEBRUIK_2017_RDNew.shp” from the “dataAmsterdam” folder onto the **Layers** and generate a new model “DistanceA”.

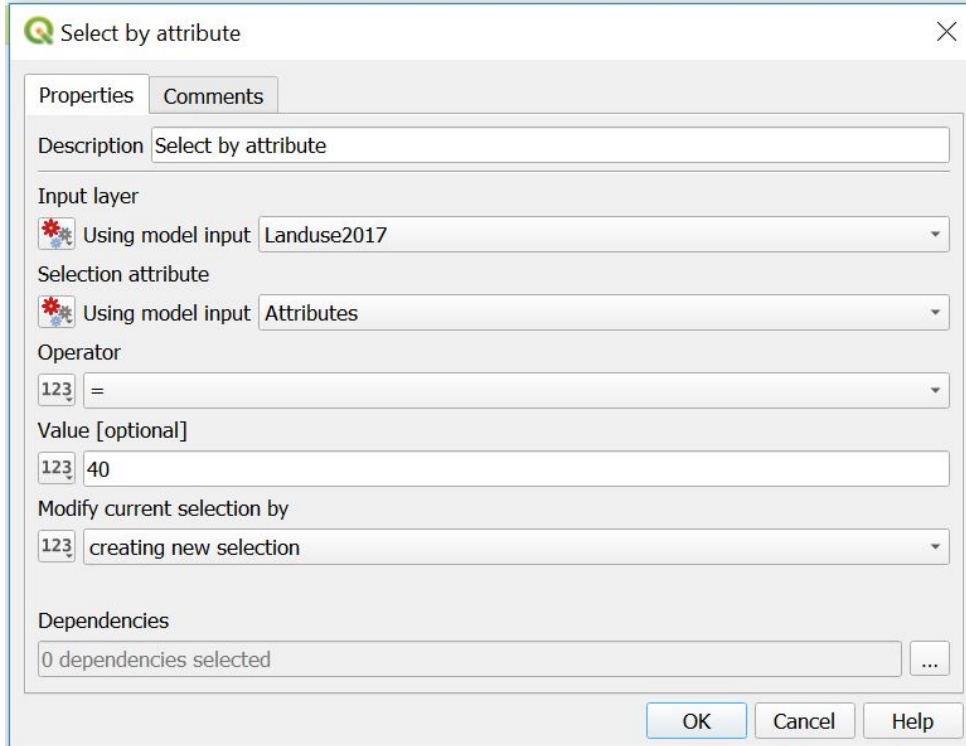
Selection of parks

In QGIS, selecting objects in a vector layer is a general way to subset data and store it in a new file. For a selection, a layer needs to exist in the **Layers**. Selections can be done by attribute (using a Python expression) or based on the location of objects. Selected objects are highlighted in yellow color, as shown in the example below. If you right click the layer and choose **Export\Save Selected Features As**, then only the selected objects can be saved as an Esri shapefile or other data formats. In the following we will make use of this same mechanism to filter out park polygons.




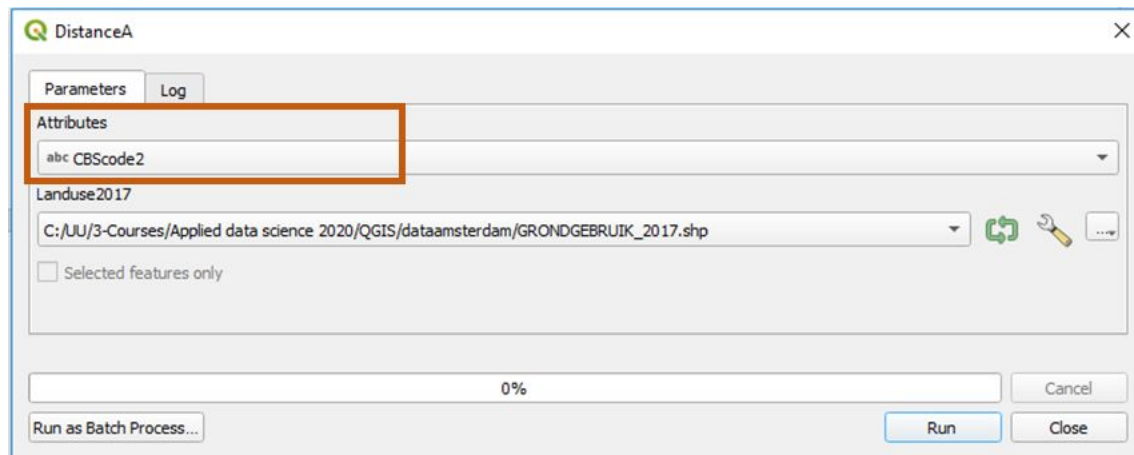
In **Model Designer**, we first add two input types to the modeler canvas. First dragging **Vector Layer** and entering “Landuse2017” under **Description**. Second input is **Vector Field** which represents an attribute table of a vector feature. Enter “Attributes” under **Description** and choose “Landuse2017” as **Parent layer**. You can define the **Allowed data type** as “String” which is the data type of the field “CBScode2”, or leave it as default.


Then add an algorithm **Select by attribute** to create new selection in the landuse data. Choose “Landuse2017” as **Input layer**. Next, change the type of **Selection attribute** into “Model Input” and select “Attributes”. The **Operator** is “=” and the **Value** is “40”.

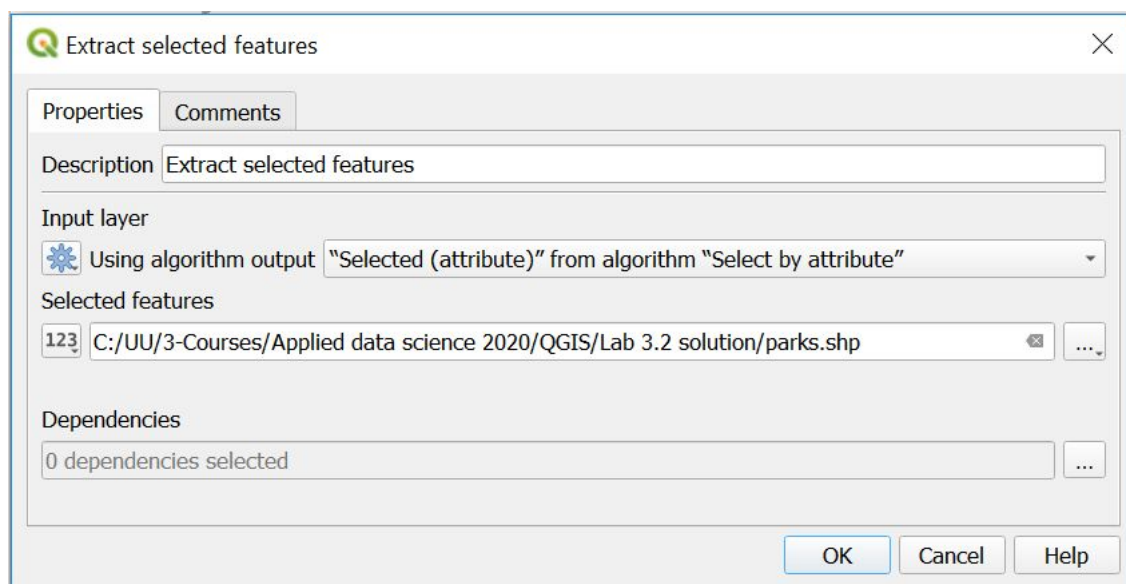


The screenshot shows the "Select by attribute" dialog box in QGIS. The "Properties" tab is selected. The "Description" field contains "Select by attribute". The "Input layer" is set to "Landuse2017". The "Selection attribute" is set to "Attributes". The "Operator" is set to "=". The "Value [optional]" field contains "40". The "Modify current selection by" dropdown is set to "creating new selection". The "Dependencies" section shows "0 dependencies selected". At the bottom, there are "OK", "Cancel", and "Help" buttons.

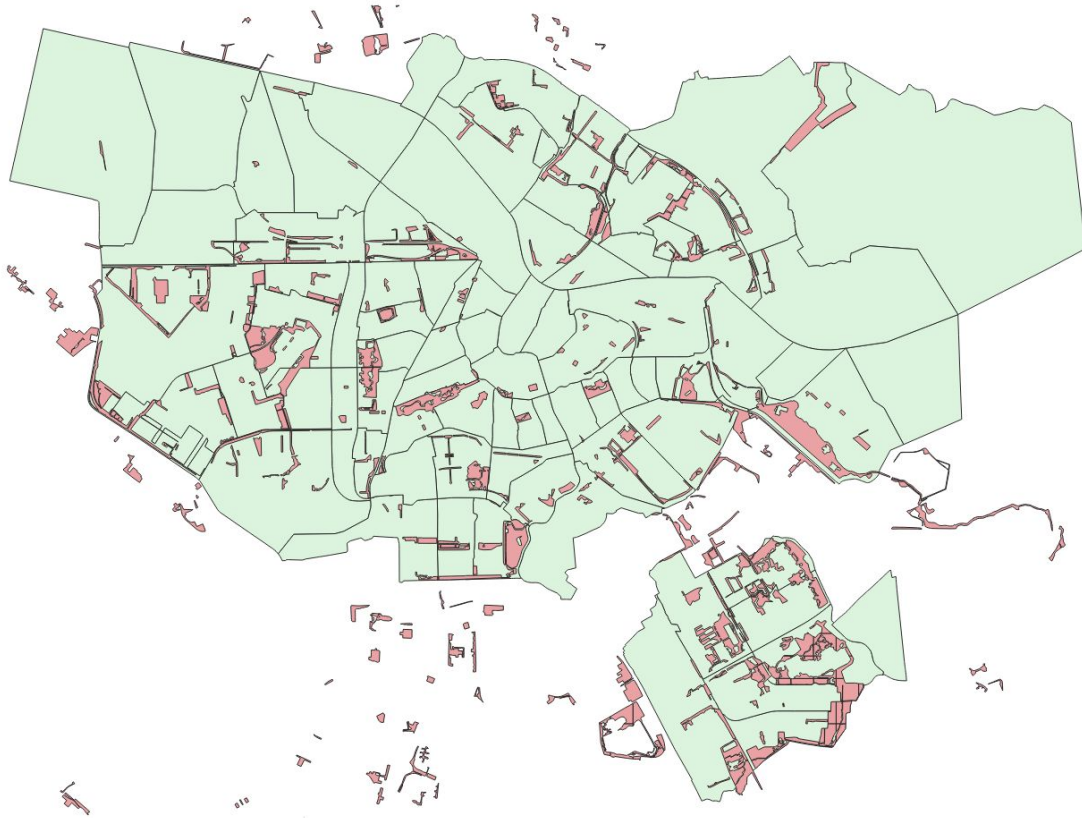
Clicking on the  button to run the model, select “CBScode2” under **Attributes** in the popped up dialog. Press **Run** and you will see the selected park areas in the “GRONDGEBRUIK_2017” layer.



In order to save the park areas as a new file, we add **Extract selected features** to the workflow. Use the output of **Select by attribute** as **Input layer**. Then change the type of **Selected features** as “Value”, click  and **Save to File** to save the output as “*parks.shp*” in the “Lad3.2 solution” folder.



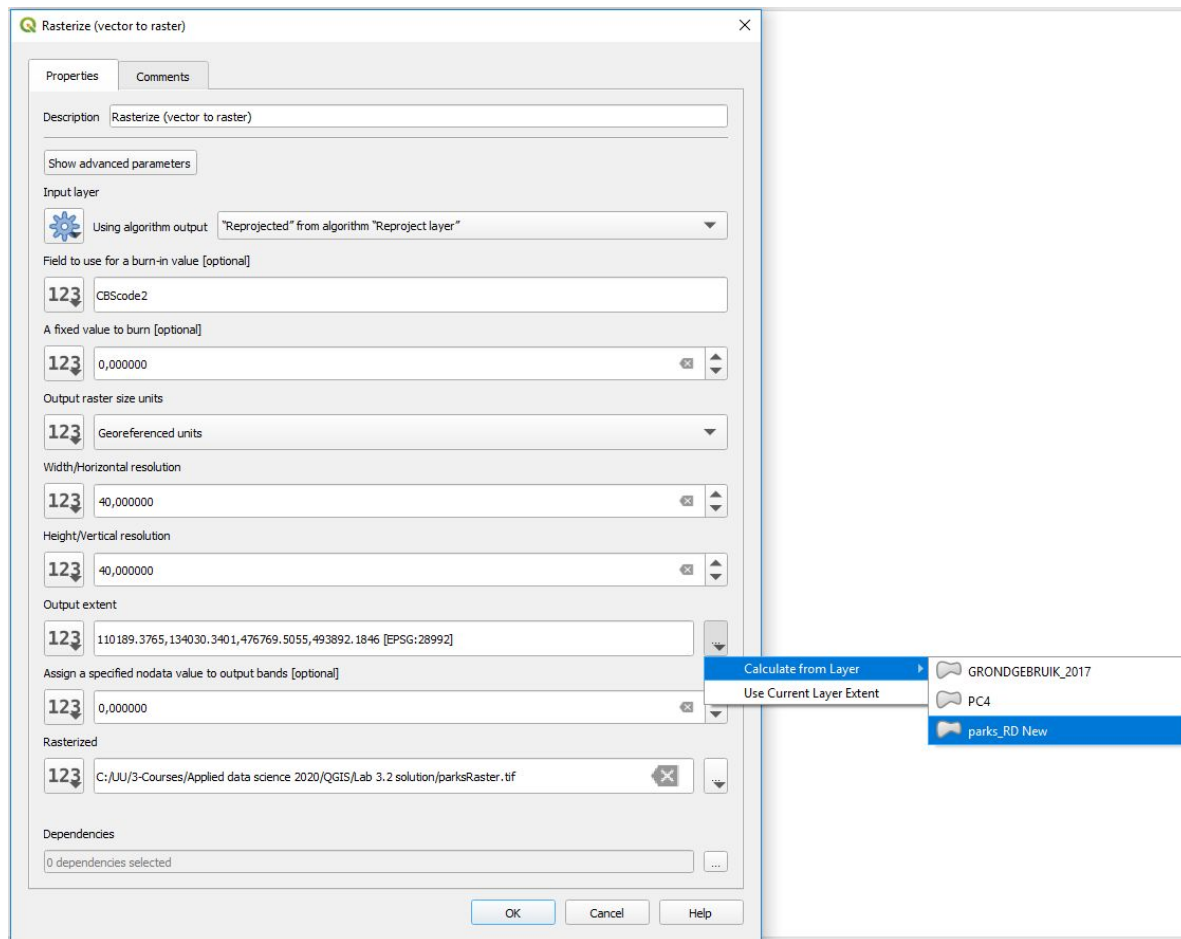
Adding the “*parks.shp*” to the **Layers**, you should see a layer of vector polygons denoting parks (here in pink):



Proximity raster

The next step is to compute a new raster layer where each cell contains the distance (= accessibility) to parks. For this purpose, you need to use **Proximity** algorithm. However, this tool only take raster data as input. Therefore you first need to rasterize the park data.

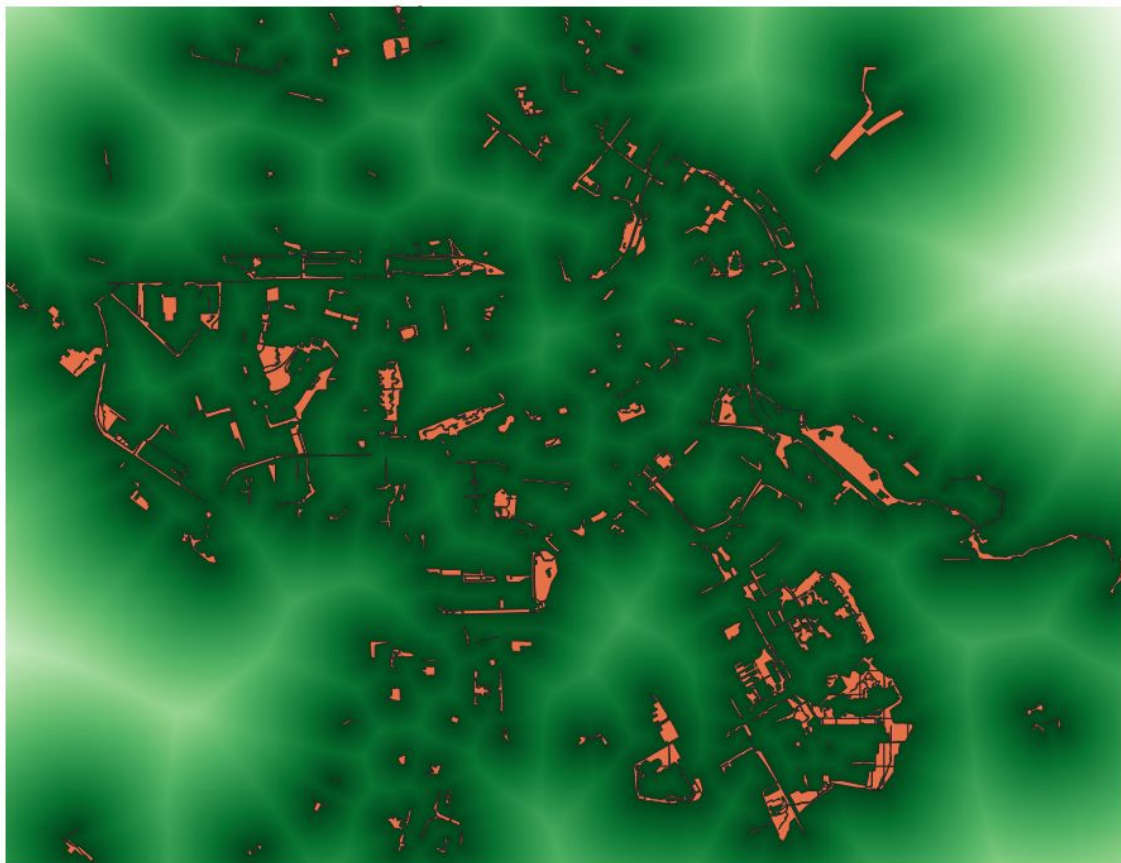
In the **Model Designer**, drag **Rasterize (vector to raster)** algorithm to the model canvas, use the output from the last step as **Input layer**. Then enter "CBScode2" under **Field to use for a burn-in value** which means using "CBScode2" value to generate pixel values in the raster. Select "Georeferenced units" (here refers to meter) under **Output raster size units**. Set the pixel size as "40" meters under **Width and Height resolution**. For the **Output extent**, you can select one layer displayed in the **Layers** panel by clicking **Calculate from Layer**. At last, save the output in the "Lab3.2 solution" folder as "*parksRaster.tif*".



You do not need to run the model now to generate the raster file. If you want to check the “parksRaster.tif” later, it should look like this:



To calculate the accessibility to parks, load **Proximity (raster distance)** algorithm, and use the output from **Rasterize** as **Input layer**. Select “Georeferenced coordinates” as **Distance units**. Save the output in “Lab3.2 solution” folder as “*dist2parks.tif*” which should look like this:



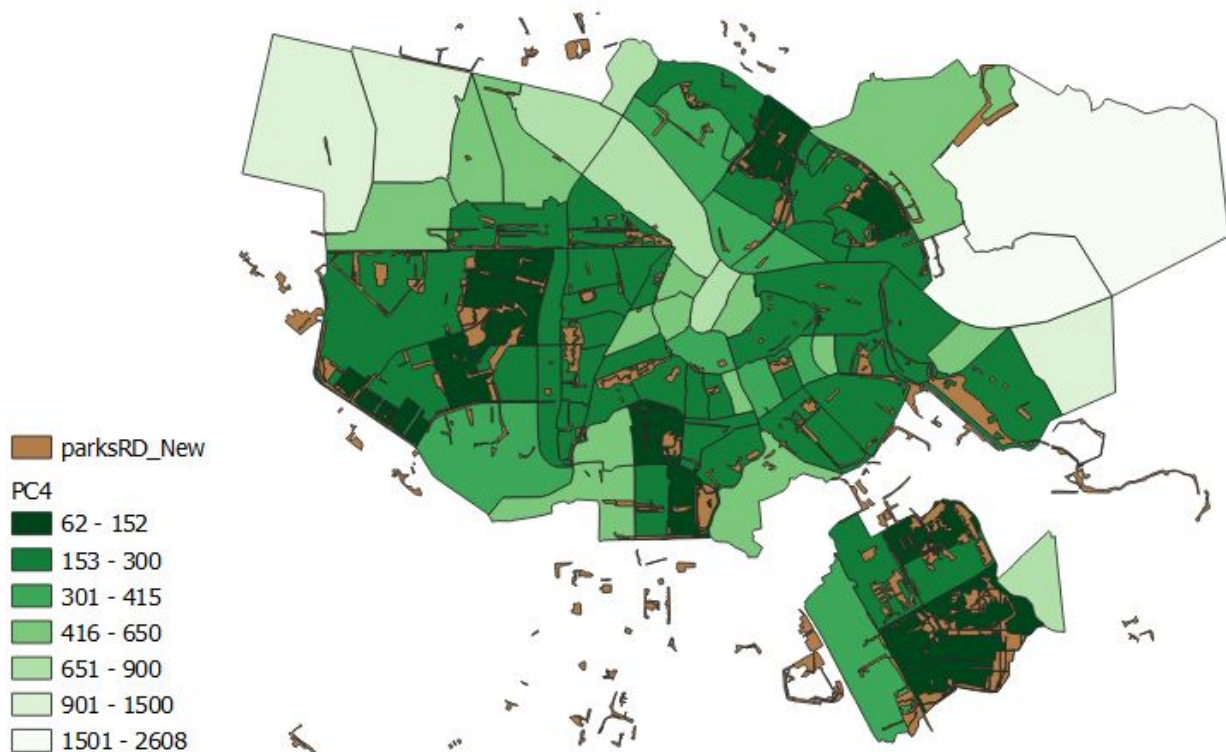
Now you have a raster file in which each pixel has a distance value to the parks. The darker the color is, the closer to the parks (here are orange polygons).

Aggregate distance raster by zonal map algebra

Finally, we use zonal map algebra again to average the distance field into “PC4” regions. Load the tool **Zonal Statistics** in the **Model Designer** and parameterize it accordingly. Make sure you select an appropriate type under **Statistics to calculate**. Enter “dist_” under **Output column prefix**.

After running the model, check the attribute table of “PC4”. The distance is stored in a new field “dist_mean” in the table.

Now visualize distance as a choropleth map by clicking **Project** → **New Print Layout**. Add map and legend. The final map should look like this:



Task 4 Density of parks in focal neighborhoods

Even though the last analysis has given us an indicator of how far you have to walk to reach a park, it does not say much about how “green” the neighbourhoods inside a postcode area are on average. This refers to the measure of how much space is covered by parks inside a neighborhood around each location inside a PC4 area. This can be measured by making use of neighbor map algebra, or a moving window average, and aggregating it with a zonal operation.

Open **QGIS Desktop 3.14 with GRASS** and create a new project “Lab3.2_Neigh” in the folder “Lab3.2 solution”. Set the CRS of the new project as “RD New”. **QGIS Desktop** cannot support a GRASS tool – **r.neighbors**, therefore we need to do the following analysis in **QGIS GRASS**.

Open the last workflow again and save it as a separate model “NeighCover”. Then delete **Proximity** and **Zonal statistics** from “NeighCover”.

Neighbor statistics

To count the number of park cells inside a 1 km² neighborhood, we need to use **r.neighbors** algorithm. However, due to function limitation of QGIS, there is not a count operation in the algorithm. As an alternative, we could use “sum” which calculate the sum of pixels within a neighborhood. Note that the pixel value of park raster is 40, therefore the sum results need to be divided by 40 to be equal to the count. There are two ways to do this:

1. First calculate the sums within neighborhoods, and then divide the results by 40 using **Field calculator** algorithm after zonal operation.
2. First change the pixel value of park raster from 40 to 1 using **r.reclass** algorithm, and then calculate the sums within neighborhoods. In this case, the sum equal to the count.


Here we do the first option. Load **r.neighbors** to the model and use the output from **Rasterize** as **Input layer**. Select “sum” under **Neighborhood operation**. Then input 25 under **Neighborhood size** to make 1km² neighborhoods (neighborhood size has to be **ODD**). Save the output in “Lab3.2 solution” folder as “*neighParks.tif*” which should look like this:



The brighter this raster gets, the more densely covered is the area with park within 1 km² rectangle around each cell.

Zonal statistics

The next step is to aggregate the raster of step 1 into PC4 areas by **Zonal Statistics** using the MEAN function. To distinguish park density output with other field, you can input “PDen_” under Output column prefix, then the output will be saved in “PDen_mean” field in “PC4.shp”. The results look like this:



	Postcode4	Opp_m2	FID_1	PDen_mean
1	1037	1467857	1	135,3846153846154
2	1041	3286912	2	241,13421550094517
3	1073	588271	3	1228,4782608695652
4	1074	431143	4	611,3559322033898
5	1072	595621	5	1310,8602150537633
6	1017	1289497	6	984,4057971014493
7	1019	3902815	7	720,7617435463394
8	1012	1136120	8	300
9	1016	790924	9	194,87889273356402
10	1013	5298489	10	948,7147335423198
11	1014	2488355	11	3277,370753323486
12	1051	982363	12	2110,1290322580644
13	1056	1032191	13	2139,0031152647975
14	1093	613883	14	1265,3369272237196
15	1094	481273	15	404,66367713004485

Now we need to use **Field Calculator** to calculate the real park density. Drag the algorithm to the model. Use the output from **Zonal statistics** as **Input layer**. Give a name for **Result field name** and select “Float” as **Field type**. Then input “PDen_mean” / 40 under **Formula**. “PDen_mean” here is the output fieldname of the last step. Save the output in “Lab3.2 solution” folder as “ParkDensity.shp”.

Field calculator

Properties Comments

Description: Field calculator

Input layer
Using algorithm output: "Zonal statistics" from algorithm "Zonal statistics"

Result field name
ParkDen

Field type
Float

Field length
10

Field precision
3

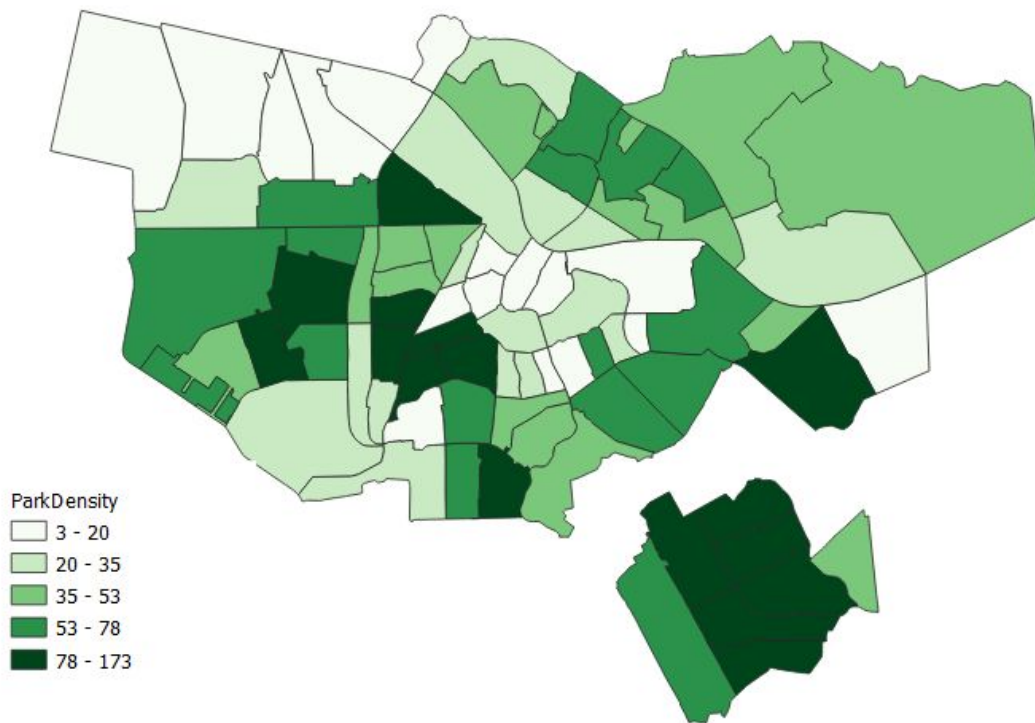
Create new field
Yes

Formula
ParkDen_mean / 40

Calculated
ParkDensity

OK Cancel Help

The final result should look like this:

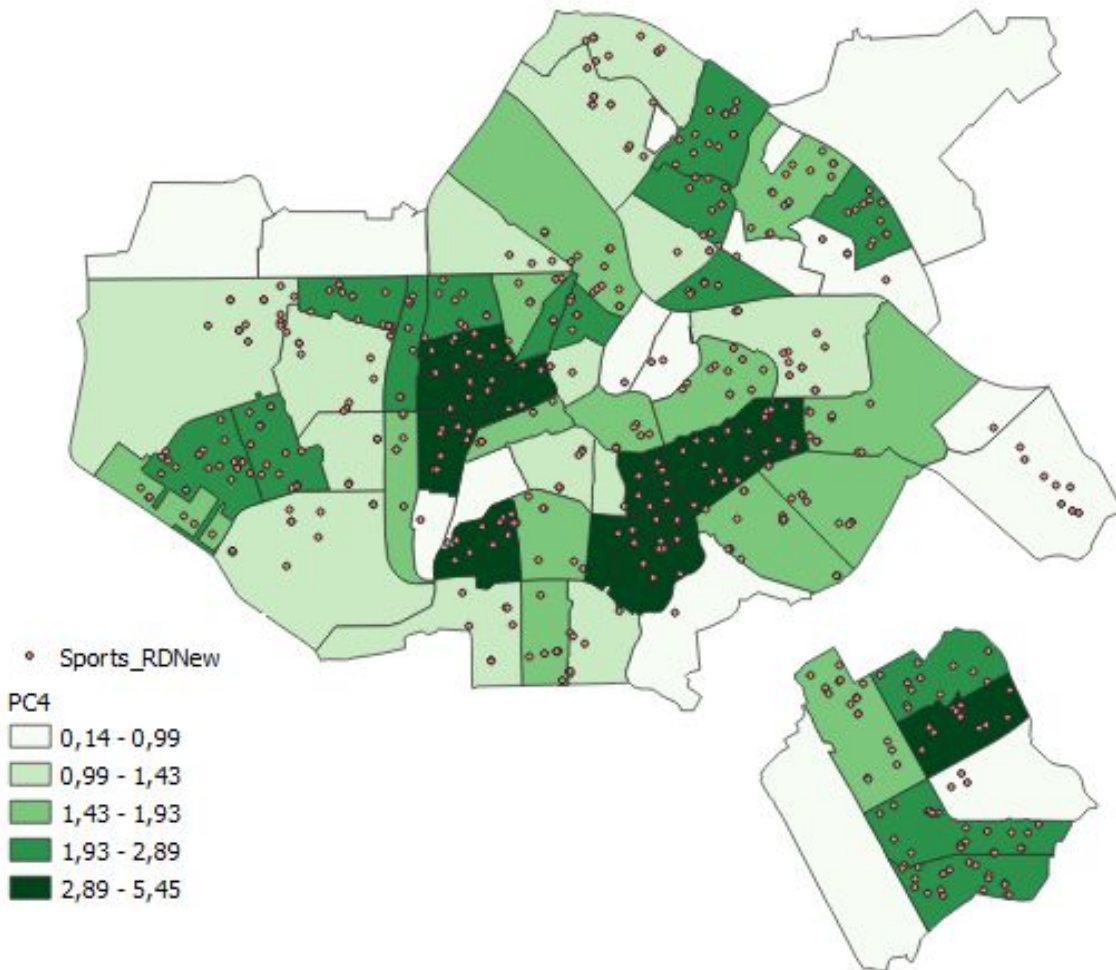


Alternative version of task 4: Computing the density of point data

This is an optional task. You don't have to submit anything for this.

To compute the density of sport facilities (file "*SPORT_OPENBAAR_RDNew.shp*") around each cell and aggregate this density into PC4 regions. For this purpose, you need to make use of a vector version of the neighbor map algebra tool, which is called **Heatmap (Kernel Density Estimation)**, using a circle radius of 1 km² (radius = 564 meter). Note that only the projected data has unit as "meter". The other parts of this task are analogous to above.

Finally, here is the result of aggregating this density raster into PC4 regions:



Assignment: Analyze new datasource and document workflows of Lab 3.1 and 3.2

There are four tasks in Lab 3.1 and 3.2. Each task introduces one set of QGIS tools for solving a specific geo-analytic problem, such as vector overlay tools: Identity and Dissolve. As an assignment, reuse one or more of these workflows to analyze another geodata source you collected in previous labs.

Finally, document your analysis by submitting the following files for Task 1,2,3,4 + your external source:

- **python scripts** exported from the Model Designer
- **model figures** (pdf or image) exported from the Model Designer
- **final maps** with clear legends

Optional Exercise: Python for GIS Analysis

This exercise is to generate a standalone Python program which makes the different analysis and aggregation workflows that were generated in the Task 1,2,3,4 (liveability of Amsterdam for older persons) reusable as functions for different kinds of input data and different parameters. The goal is to be able to automatically vary the input for complex spatial analyses. This will allow you to simultaneously try out many alternatives in your analysis.

For this purpose, you should make use of the model designer models that you generated and produce runnable Python functions from them which abstract from the data sources and parameters used in these models. Finally you should iteratively apply these Python functions to different input data from the Amsterdam Data Lab.

The learning goals of this exercise are:

- Learn to set up a Python programming environment in QGIS
- Learn to program reusable Python functions with customizable input and output variables
- Learn to use certain PyQGIS methods and classes and their syntax which correspond to QGIS tools
- Learn to customize data and parameter value inputs using string concatenation
- Learn some simple programming flow logic

Regarding the Python syntax, what you need to know in order complete this assignment involves:

- How to write and concatenate file paths in Python
- How Python uses and assigns variables
- The Python syntax for defining and calling functions
- The Python syntax for calling class methods
- A good understanding of the syntax of some PyQGIS methods

Download “PythonEx” folder which contains:

- Four Python scripts exported from models of Task 1,2,3,4
- A AggregationFunctionsTemplate.py
- A batch file for setting up QGIS Python environment in your PyCharm

Set up QGIS Python environment

Set up batch file

Open “pycharm-pyqgis.bat” file. It is a command line scrip for setting up the variables that we need for the QGIS libs, also setting the path to Pycharm. You need to update the following variables to your own:

- `QGIS_ROOT`: this is the path to your *QGIS installation folder*.
- `QGISNAME`: here is the QGIS name under your *QGIS installation folder > apps*. if you are using QGIS in long term release version, then here is *qgis-ltr*.
- `QTNAME`: here is the Qt name under your *QGIS installation folder > apps*.
- `PYCHARM`: this is the path to your Pycharm application.

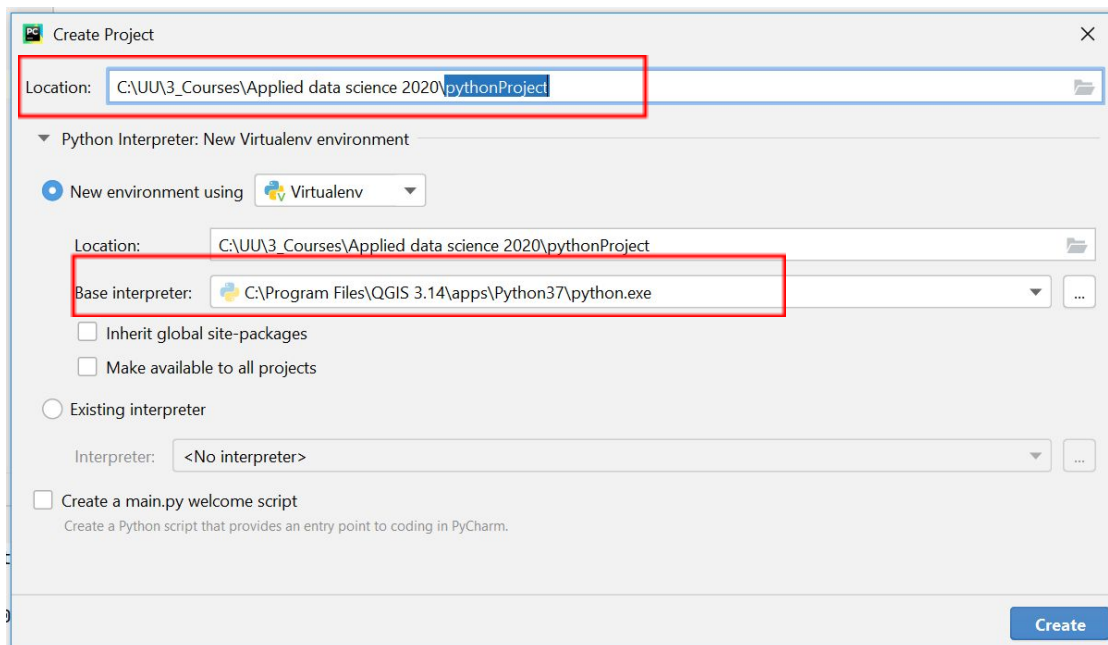
Save the batch file and then double click it. It should open Pycharm automatically.

New python project

Click **New Project** in Pycharm, give a directory to save your Python code under “Location”.

Then specify the Python Interpreter by clicking “...” and select the Python.exe *from your QGIS install folder*.

Finally, click “Create”.



To check whether your Pycharm can interpret QGIS packages, type into Python Console:

```
>>> from qgis.core import *
```

If there is no error, then the QGIS environment is set up correctly.

Generate a script with reusable functions for aggregation analysis

Open the file “AggregationFunctionsTemplate.py” in your Pycharm.

The file begins with a commented line informing about the author. You can put your name there if you like.

```

#-----
# Name:      Aggregation functions in QGIS
# Purpose:   Implement different spatial analysis, overlay and data aggregation functions in Python using PyQGIS
#
# Author:    Simon Scheider, Haiqi
#
# Created:   8/11/2020
# With QGIS : 3.14.15
#-----

```

Then it indicates the relevant Python modules which are imported for this program to run: `os`, `sys`, `qgis.core`, `QgsNativeAlgorithms`.

To avoid errors such as “invalid layer” in your code, you should set your QGIS prefix path which specify the QGIS resources such as data providers. Update your QGIS path in

```
QgsApplication.setPrefixPath
```

```

#See https://gis.stackexchange.com/a/155852/4972 for details about the prefix
QgsApplication.setPrefixPath("C:/Program Files/QGIS 3.14/apps/qgis", True)
qgs = QgsApplication([], False)
qgs.initQgis()

```

Next, it append the path where QGIS processing plugin can be found. We also need `QgsNativeAlgorithms` to be able to use tools such as “intersection” in the program.

```

# Append the path where processing plugin can be found
sys.path.append(r'C:\Program Files\QGIS 3.14\apps\qgis\python\plugins')
import processing # In order to call algorithms
from processing.core.Processing import Processing
Processing.initialize()
QgsApplication.processingRegistry().addProvider(QgsNativeAlgorithms())

```

As the first step to generate reusable Python code for the aggregation workflows, you need to update your workspace to the local folder in which you put your Amsterdam data, and define the paths to the standard source data that you used in the previous tasks.

```

#Set directory, CRS
amsterdamfolder = r'C:\UU\3_Courses\Applied data science 2020\dataAmsterdam'
#crs = QgsCoordinateReferenceSystem("EPSG:28992")

#Standard source data
pc4layer = QgsVectorLayer(os.path.join(amsterdamfolder, 'PC4.shp'), 'PC4', 'ogr')
cbslayer = QgsVectorLayer(os.path.join(amsterdamfolder, 'CBS_Buurt_2014_AP_RDNew.shp'), 'CBS', 'ogr')
resultfolder = r'C:\UU\3_Courses\Applied data science 2020\PythonEx\ArealInterp'

```

‘ogr’ is one of the data providers in QGIS, which including the file formats for many GIS systems, database formats, and web services. It is always a good idea to put long strings (like

paths) into a variable, as this makes it easier to reuse and read. Assign your path to `resultfolder` where you can put all your output.

Program a function for areal interpolation from CBS data

You have exported your Python scripts from previous Tasks in Lab 3.1 and 3.2. Now, open the one from the areal interpolation model (Task1) in Notepad or Notepad++ or open “ArealInter.py” from PythonEx folder.

In `AggregationFunctionsTemplate.py`, an example for this function is already in the code:

```
# Function for Task1 workflow - Areal Interpolation
def arealInterpol(targetfile=pc4layer, overlayfile=cbslayer, outputfile='final.shp'):
```

Note also that it has the following input variables, which come with a default value:

- *targetfile* = the file containing the regions that are the targets for aggregation). In the previous task, this was the PC4 shapefile, but it could be any other kind of region file.
- *overlayfile* = the file containing the field “P_65_EO_JR” that should be interpolated. In the previous task, this was the CBS shapefile.
- *outputfile* = the name of the feature class that should be produced as result of the interpolation procedure

We need to clarify the tools used in ArealInterpolation models. The first tool is *intersection*. You can copy the corresponding codes from “ArealInter.py” (as shown below) and paste them inside the function.

```
# Intersection
alg_params = {
    'INPUT': parameters['TargetLayer'],
    'INPUT_FIELDS': [],
    'OUTPUT': 'C:/UU/3_Courses/Applied data science 2020/Lab3.1/Lab3.1
solution/area_interpolation_data/clipped.shp',
    'OVERLAY': parameters['JoinLayer'],
    'OVERLAY_FIELDS': [],
    'OVERLAY_FIELDS_PREFIX': '',
    'OUTPUT': QgsProcessing.TEMPORARY_OUTPUT
}
outputs['Intersection'] = processing.run('native:intersection', alg_params,
context=context, feedback=feedback, is_child_algorithm=True)

feedback.setCurrentStep(1)
if feedback.isCanceled():
    return {}
```

Now we need to adjust the different PyQGIS method calls as below so that they make use of these variables.

```
alg_params = {
    'INPUT': targetfile,
    'INPUT_FIELDS': [''],
    'OUTPUT': os.path.join(resultfolder, 'clipped.shp'),
    'OVERLAY': overlayfile,
    'OVERLAY_FIELDS': [''],
    'OVERLAY_FIELDS_PREFIX': '',
}
clipped = processing.run('native:intersection', alg_params)
```

`alg_params(...)` give all the necessary parameters for intersection tool. `processing.run(...)` actually executes the *intersection* operation.

Inside `alg_params(...)`, note that the `'INPUT'` of this method is the targetfile, and `'OVERLAY'` is the overlayfile. For `'OUTPUT'`, you can define the location and output name using `os.path.join()`.

Inside `processing.run(...)`, the first parameter is the tool index which refer to intersection from a provider: `QgsNativeAlgorithms`. `clipped` is a dictionary which save the result.

You do not need `'OUTPUT': QgsProcessing.TEMPORARY_OUTPUT` from “ArealInter.py”. Because it means the output is a temporary file which cannot be saved in the local folder. The feedback codes is for a multi-step process and report feedback of each step. You can delete them too.

Next is the parametrization of the *Add geometry attributes* method. Same as the previous step, copy the corresponding codes from “ArealInter.py” and adjust the variables:

```
alg_params = {
    'CALC_METHOD': 0,
    'INPUT': clipped['OUTPUT'],
    'OUTPUT': os.path.join(resultfolder, 'add_geom.shp'),
}
add_geom = processing.run('qgis:exportaddgeometrycolumns', alg_params)
```

Note that the input of this method is the output `clipped` of the last step. `clipped['OUTPUT']` gives the local directory of “clipped.shp”.

Do the similar adaption for the other methods calls. Each of them requires correct input and output parameter.

At the end of the function, remember to return to the final result of the function. We can also print a message about its success before returning the result file:

```
print("Interpolated " + overlayfile.name() + "'s attribute " + 'P_65_EO_JR' +
      " into " + targetfile.name())
return result
```

To make use of the function, scroll down to the `Main()` method and call `arealInterp(...)`.

```
def main():
    arealInterpol(targetfile=pc4layer, overlayfile=cbslayer,
outputfile='final.shp')
```

Now you can run the entire script and generate all the output files for areal interpolation in your local result folder.

You may receive Warnings such as:

```
Warning 1: Value 18669377.6700000018 of field product of feature 80 not
successfully written. Possibly due to too larger number with respect to field
width
```

This warning will not lead errors to your results, but just remind you that the product field is not long enough. The warning is caused by the add geometry attribute which generates the “area” field of 23 length. However, the length of product field is set to 10 in the workflow. To avoid this warning, you can adjust the length of product as 23 in the Field calculator method call in the code (as shown below), as well as the length of area and product in Aggregate method call.

```
alg_params = {
    'FIELD_LENGTH': 23, # set length of product field
    'FIELD_NAME': 'product',
    'FIELD_PRECISION': 3,
    'FIELD_TYPE': 0,
    'FORMULA': 'CASE \r\nWHEN attribute($currentfeature, \'BU_CODE\') IS NULL
OR attribute($currentfeature, \'P_65_EO_JR\') <= 0 THEN NULL ELSE
attribute($currentfeature, \'area\') * attribute($currentfeature,
\'P_65_EO_JR\')\r\nEND',
    'INPUT': add_geom['OUTPUT'],
    'NEW_FIELD': True,
    'OUTPUT': os.path.join(resultfolder, 'calc.shp'),
}
calc = processing.run('qgis:fieldcalculator', alg_params)
```

Run the entire script again. If no errors are produced, the console output should look like this:

```
Interpolated CBS's attribute P_65_EO_JR into PC4
```

```
Process finished with exit code 0
```

Optional exercise: Python for other GIS analysis workflow

We have introduced how to generate a standalone Python script based on the Python file exported from Task 1 workflow. As an exercise, you could fill in other functions in “AggregationFunctionsTemplate.py”, or generate a function for your own analysis and data.

Note that you need to add corresponding method calls to the *main()* method which makes use of these functions. You do not need to submit anything, but it is good to try out.