Hans Alberto Franke

Date: 20/11/2020

2.1 Querying datasets

Each task is comment in the python code inside the scrip file: explore_bag.py.

Task: Print the number of layers included in the dataset.

```
Task1: Layers 3
```

Task: Print for each layer the layer name and CRS.

```
Layer: 0 Pand
CRS PROJCS["Amersfoort / RD New",
    GEOGCS["Amersfoort",
        DATUM["Amersfoort",
            SPHEROID["Bessel 1841",6377397.155,299.152
                AUTHORITY["EPSG","7004"]],
            AUTHORITY["EPSG","6289"]],
        PRIMEM["Greenwich",0,
            AUTHORITY["EPSG","8901"]],
        UNIT["degree",0.0174532925199433,
            AUTHORITY["EPSG","9122"]],
        AUTHORITY["EPSG","4289"]],
    PROJECTION["Oblique_Stereographic"],
    PARAMETER["latitude_of_origin",52.1561605555556],
    PARAMETER["central_meridian",5.38763888888889],
    PARAMETER["scale_factor",0.9999079],
    PARAMETER["false_easting",155000],
    PARAMETER["false_northing",463000],
    UNIT["metre",1,
        AUTHORITY["EPSG","9001"]],
    AXIS["Easting",EAST],
    AXIS["Northing",NORTH],
    AUTHORITY["EPSG","28992"]]
Layer: 1 Verblijfsobject
CRS PROJCS["Amersfoort / RD New",
    GEOGCS["Amersfoort",
        DATUM["Amersfoort",
            SPHEROID["Bessel 1841",6377397.155,299.1528128,
                AUTHORITY["EPSG","7004"]],
            AUTHORITY["EPSG","6289"]],
        PRIMEM["Greenwich",0,
            AUTHORITY["EPSG","8901"]],
        UNIT["degree",0.0174532925199433,
            AUTHORITY["EPSG","9122"]],
        AUTHORITY["EPSG","4289"]],
    PROJECTION["Oblique_Stereographic"],
    PARAMETER["latitude_of_origin",52.1561605555556],
    PARAMETER["central_meridian",5.38763888888889],
    PARAMETER["scale_factor",0.9999079],
    PARAMETER["false_easting",155000],
    PARAMETER["false_northing",463000],
    UNIT["metre",1,
        AUTHORITY["EPSG","9001"]],
    AXIS["Easting",EAST],
    AXIS["Northing",NORTH],
    AUTHORITY["EPSG","28992"]]
Layer: 2 Wijken
CRS PROJCS["Amersfoort / RD New",
    GEOGCS["Amersfoort",
        DATUM["Amersfoort",
            SPHEROID["Bessel 1841",6377397.155,299.1528128,
                AUTHORITY["EPSG","7004"]],
            AUTHORITY["EPSG","6289"]],
        PRIMEM["Greenwich",0,
            AUTHORITY["EPSG","8901"]],
        UNIT["degree",0.0174532925199433,
            AUTHORITY["EPSG","9122"]],
        AUTHORITY["EPSG","4289"]],
    PROJECTION["Oblique_Stereographic"],
    PARAMETER["latitude_of_origin",52.1561605555556],
    PARAMETER["central_meridian",5.38763888888889],
    PARAMETER["scale_factor",0.9999079],
    PARAMETER["false_easting",155000],
    PARAMETER["false_northing",463000],
    UNIT["metre",1,
        AUTHORITY["EPSG","9001"]],
    AXIS["Easting",EAST],
    AXIS["Northing",NORTH],
    AUTHORITY["EPSG","28992"]]
```

Task: Print the number of features in the layer.

```
Task3: Layer name Verblijfsobject
Task3: Features 496420
```

Task: Print the name and type of each field in the layer.

```
Task4: Number of fields  2
Name of field: oppervlakte
Type of field Integer
Name of field: gebruiksdoel
Type of field String
```

Question: What is the total surface area given in the location layer?

```
Total area: 59255192
```

Question: What is the coordinate of the feature with the index 439774?

```
POINT (121815.991 487912.647 0)
X 121815.991
Y 487912.647
```

2.2 Obtaining building properties

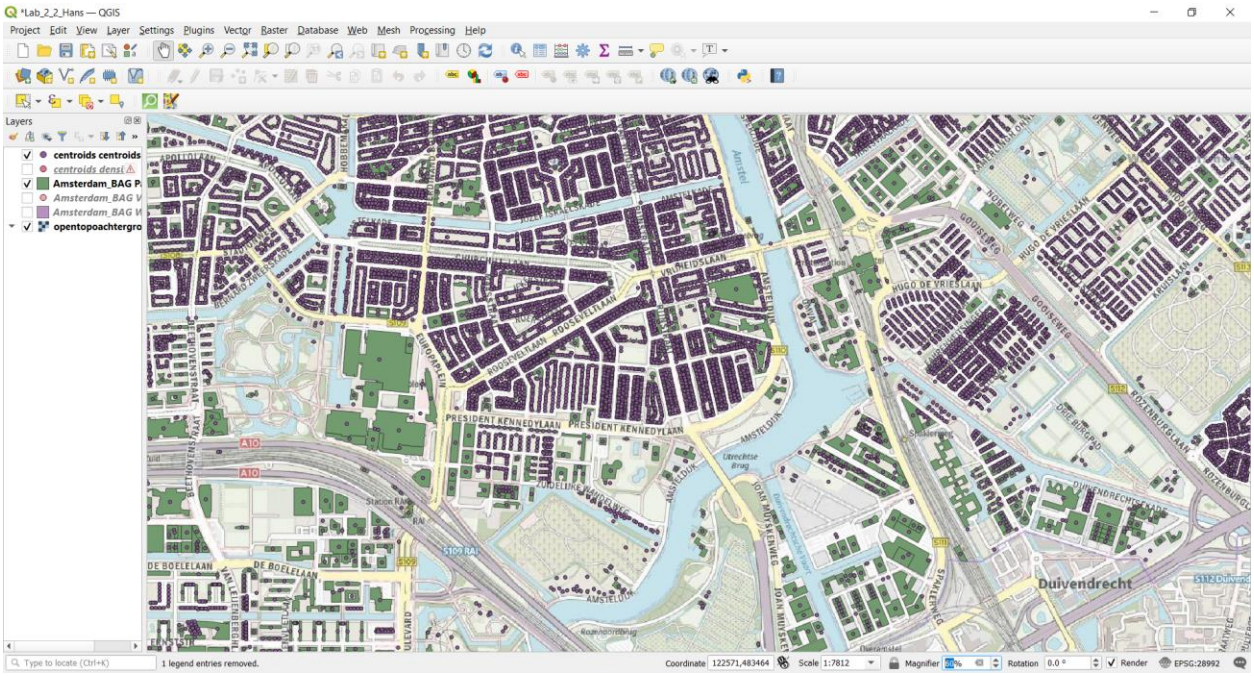
Each task and question is commented in the python code. Filename: building_surface_area.py

Task: Add a field area to your layer centroids

```
#TASK: To add a new field holding floating point values to a layer use:
field = ogr.FieldDefn("area", ogr.OFTReal)
centroid_layer.CreateField(field)
```

Task: Calculate the area and the centroid location of each building.

Final map: purple points are the centroids, and the green polygons represents de buildings. The map was zoom in to see the both representations. The background of the map is the PDOK map of Netherlands. CRS = EPSG: 28992.



2.3 Computing building densities

Question: What is the density of the district with feature id 54 (Museumkwartier)?

Looking in QGIS to check!

centroids density — Features Total: 99, Filtered: 99, Selected: 0

	fid	name	density	fraction
52	52	Schinkelbuurt	2307.16458218...	31.9105830610...
53	53	Willemspark	1797.03204233...	19.4907761050...
54	54	Museumkwartier	2117.07447604...	28.4837082698...

Script in Python to get the same results as GIS:

```
14
15  ## Question: What is the density of the district with feature id 54 (Museumkwartier)?
16
17
18  c_db = ogr.GetDriverByName('GPKG').Open("centroids.gpkg", update=0)
19  density = c_db.GetLayerByName('density')
20  density.GetFeature(54).GetField("density")
21
22
```

3.1 Retrieving school data

File: get_school_data.py

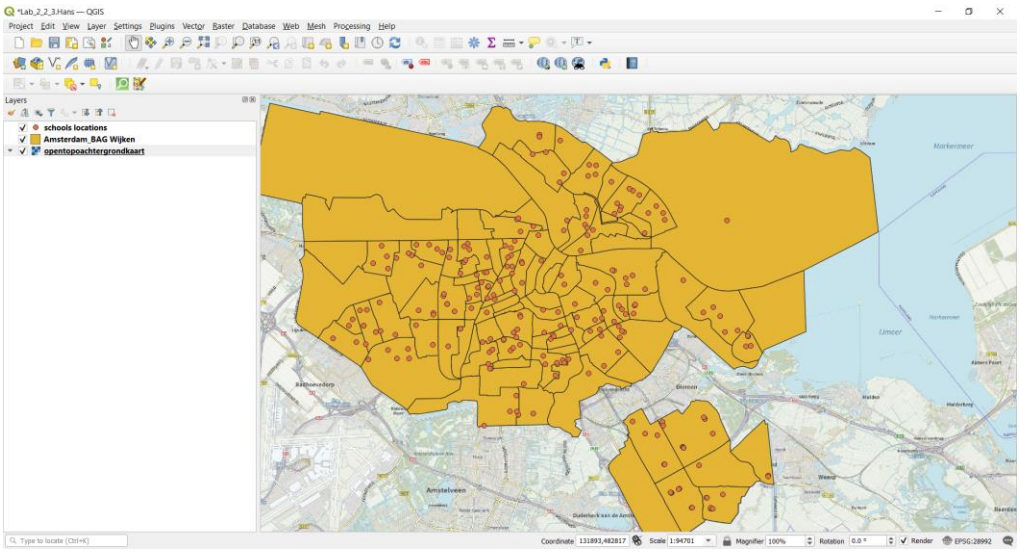
Request Parameters: PO = Primary Schools

```
# -*- coding: utf-8 -*-
"""
Created on Fri Nov 20 17:09:46 2020

@author: hansf
"""

import requests

#query on dataset where po = primary schools
r = requests.get('https://schoolwijzer.amsterdam.nl/nl/api/v1/lijst/po/', verify = False)
print(r.json())

#save file to disk
import json
with open('data.json', 'w', encoding='utf-8') as f:
    json.dump(r.json(), f, ensure_ascii=False, indent=4)
```

3.2 Constructing point geometries

Task: Complete your Python script to create the school layer and run it.

Map with the results. Pink points = school locations. Yellow polygon = districts. Background = PDOK of Amsterdam. CRS = EPSG 28992



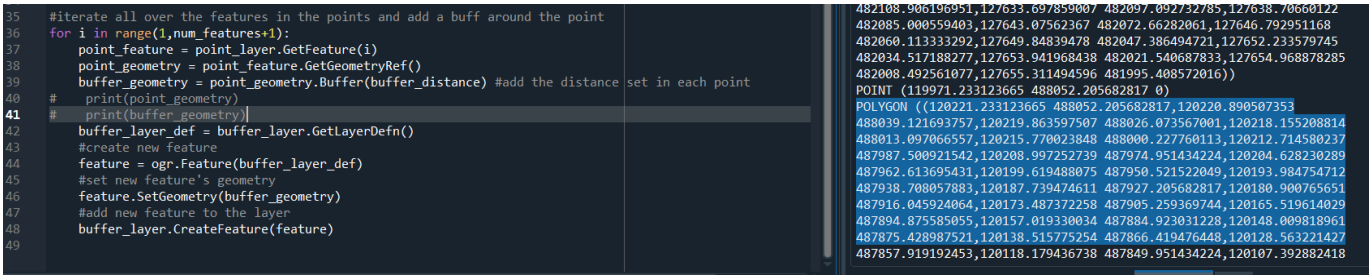
3.3 Adding buffer areas around schools

Question: What is the geometry type of the layer buffer?

I chose polygons:

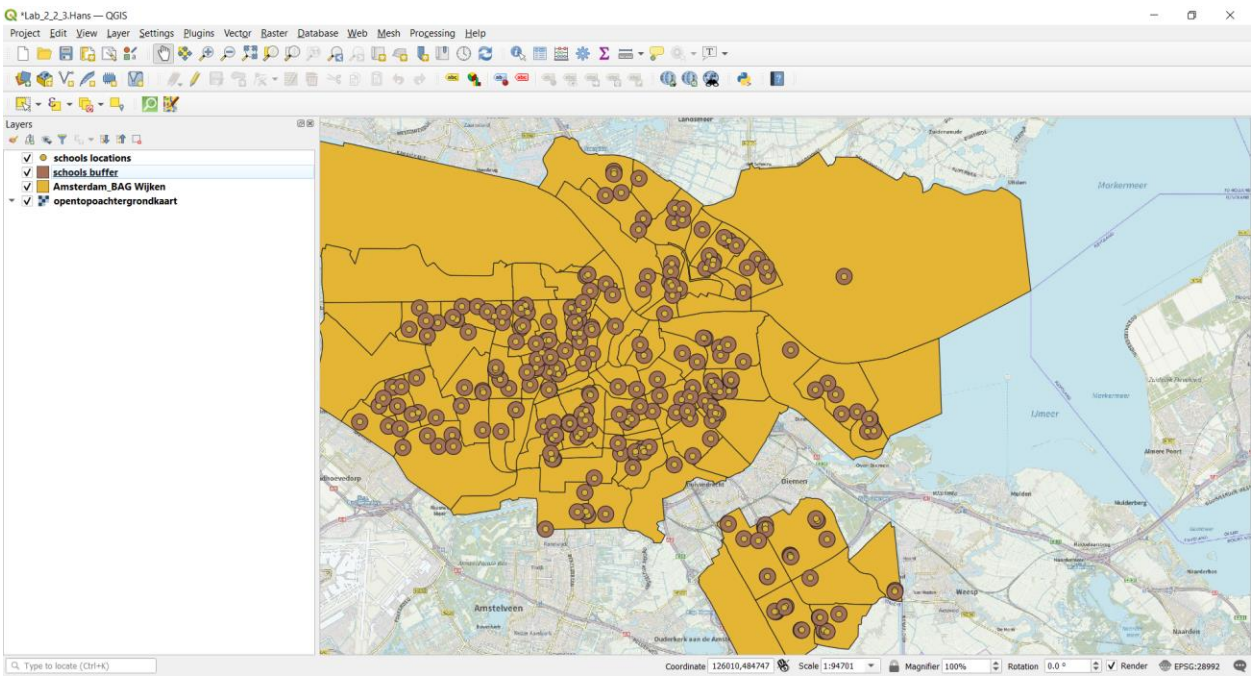
```
19
20  buffer_layer = data_source.CreateLayer('buffer', srs=rdNew, geom_type=ogr.wkbPolygon)
```

And confirmed in python code, when you add a buffer_distance to a point it becomes a polygon.



Task: Create new features for the buffer geometries and add them to the buffer layer.

You can see that the points have a brown “circle” outside, this is the buffer area.



3.4 Merging geometries

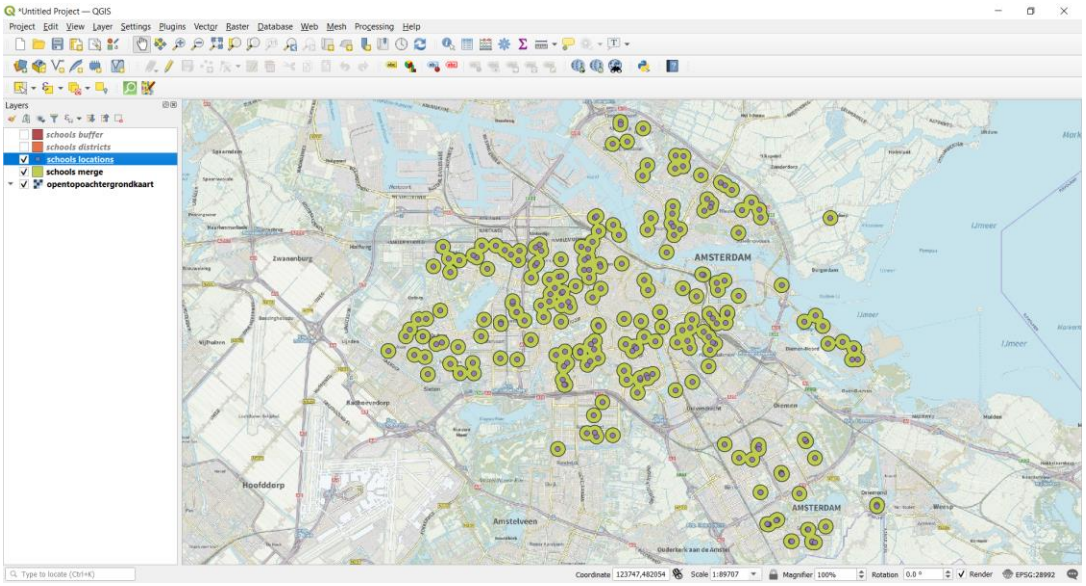
Question: What is the geometry type of the layer merge?

MultiPolygon, because it will store a sum of many polygons (the buffers)

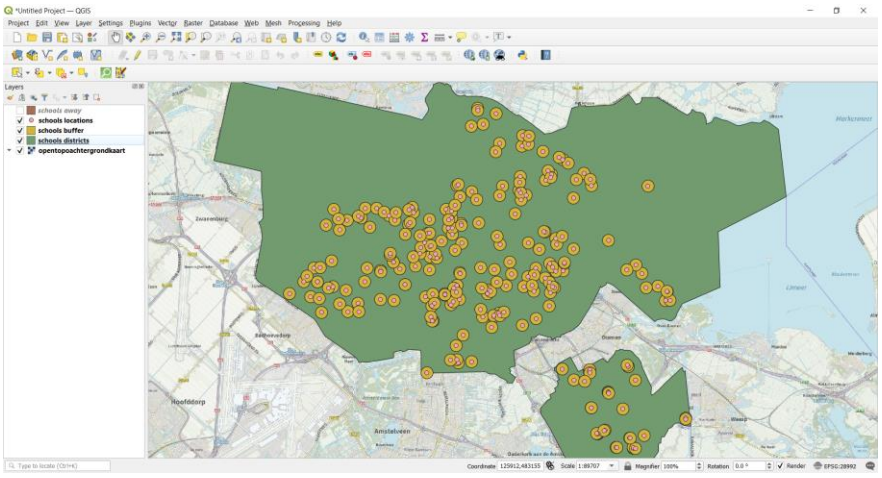
```
merge_layer = data_source.CreateLayer('merge', srs=rdNew, geom_type=ogr.wkbMultiPolygon)
```

Showing the map in QGIS:

Now you see the buffers merged as a multipolygon.



Task: Create a second script merge_districts.py to merge the districts from the Wijken input layer of the Amsterdam_BAG.gpkg dataset. Merge the district geometries to one new geometry and add the result to the schools. gpkg dataset as the new layer districts.



3.5 Compute area far from schools

Question: Which operation will you use to compute the area far away from schools?

d) Difference

Python Code:

districts_layer.SymDifference(merge_layer, away)

Returns an object that contains every point in either g1 or g2 but not the points that are in both g1 and g2. g1 and g2 should have the same dimension. Equivalent to the UNION of the DIFFERENCEs of g1, g2 and g2, g1.

Question: What is the size of the area considered as far away from public schools?

The area considered far away from schools is (m2): 186648064.65494493

Appendix: Scripts

Filename: LAB_2_2_Scripts_HansFranke.rar

LAB_2_2_scripts_Hans Franke.rar

Below there is the code paste for each file.

Explore_bag.py

```python
# -*- coding: utf-8 -*-
from osgeo import gdal, ogr
filename = 'Amsterdam_BAG.gpkg'
data_source = ogr.GetDriverByName('GPKG').Open(filename, update=0)

#Task1: Print the number of Layers Included in Dataset:
print("Task1: Layers",data_source.GetLayerCount())

#number of layers
num_layers = data_source.GetLayerCount()

#Task2: Print for each layer the layer name and CRS
for i in range(0,num_layers):
    layer = data_source.GetLayerByIndex(i)
    srs = layer.GetSpatialRef()

    print("Layer:", i, layer.GetName())
    print("CRS", srs)
    print("\n")

print("End TASK \n\n\n")

buildings = data_source.GetLayerByName('Verblijfsobject')

#task3: Print the number of Features in the layer
print("Task3: Layer name", buildings.GetName())
print("Task3: Features", buildings.GetFeatureCount())
locations_def = buildings.GetLayerDefn()
num_fields = locations_def.GetFieldCount()
print("Task4: Number of fields ",num_fields)

#task 4: print the name and type of each field in the layer
for i in range(0,num_fields):
    print("Name of field:",locations_def.GetFieldDefn(i).GetName())
    print("Type of field", locations_def.GetFieldDefn(i).GetTypeName())

#task5: Add code to your script that iterates over all features in the layer, retrieves the value of the field oppervlakte
#(surface area) and adds up the area.
# oppervlakte = "Surface"

# Question: What is the total surface area given in the location layer?

num_features = buildings.GetFeatureCount()
print("Number of features:", num_features)

field_name = "oppervlakte"
area = 0
for i in range(1,num_features+1):
    feature = buildings.GetFeature(i)
    field_value = feature.GetField(field_name)
    area = area + field_value
print("Total area:", area)

#Question: What is the coordinate of the feature with the index 439774?
feature = buildings.GetFeature(439774)
geometry = feature.GetGeometryRef()
print(geometry)
print("X", geometry.GetX())
print("Y", geometry.GetY())
```

Building_surface_areas.py

```python
#2.2 Obtaining building properties

from osgeo import gdal, ogr
filename = 'Amsterdam_BAG.gpkg'
data_source = ogr.GetDriverByName('GPKG').Open(filename, update=0)
#number of layers
num_layers = data_source.GetLayerCount()
print(num_layers)
#Load layer
layer_original = data_source.GetLayerByName("Pand")
#check
print("Layer:", layer_original.GetName())

from osgeo.osr import SpatialReference
rdNew = SpatialReference()
rdNew.ImportFromEPSG(28992)

#Create the new dataset with the output layer centroids using a point geometry type as:
centroid_source = ogr.GetDriverByName('GPKG').CreateDataSource('centroids.gpkg')

#Import layer
centroid_layer = centroid_source.CreateLayer('centroids', srs=rdNew, geom_type=ogr.wkbPoint)

#TASK: To add a new field holding floating point values to a layer use:
field = ogr.FieldDefn("area", ogr.OFTReal)
centroid_layer.CreateField(field)

#loading definitions
centroid_layer_def = centroid_layer.GetLayerDefn()
print(centroid_layer_def)

#TASK:Calculate the area and the centroid location of each building.
#You can then iterate over each feature in a layer using a for loop.
num_features = layer_original.GetFeatureCount() #define the superior limit
print(num_features)

for i in range(1,num_features+1):

    #load geometry from the original layer
    feature = layer_original.GetFeature(i) #each feature is a row in the table
    house_geometry = feature.GetGeometryRef()

    point_feature = ogr.Feature(centroid_layer_def) # get the definition for the new feature
    point = ogr.Geometry(ogr.wkbPoint)

    centroid = house_geometry.Centroid() # The centroid that we want to add
    point.AddPoint(centroid.GetX(), centroid.GetY()) #  make a point from this feature
    point_feature.SetGeometry(point) # add the place of the centroid to the new point

    house_area = house_geometry.GetArea()

    point_feature.SetField('area', house_area)
    centroid_layer.CreateFeature(point_feature) #add to the layer
```

Densities.py

```python
from osgeo import gdal, ogr
from tqdm import tqdm #progress bar

filename = 'Amsterdam_BAG.gpkg'
filename2 = 'centroids.gpkg'

#Load Sources
data_source = ogr.GetDriverByName('GPKG').Open(filename, update=0)
data_source2 = ogr.GetDriverByName('GPKG').Open(filename2, update=1)

#Load layers
layer_wijken = data_source.GetLayerByName("Wijken") #Wijken = Neighbordhoods
layer_centroids = data_source2.GetLayerByName("centroids")
density =  data_source2.GetLayerByName("density")

from osgeo.osr import SpatialReference
rdNew = SpatialReference()
rdNew.ImportFromEPSG(28992)

#create a new layer
if data_source2.GetLayerByName("density"):
    data_source2.DeleteLayer("density")
density_layer = data_source2.CreateLayer('density', srs=rdNew, geom_type=ogr.wkbPoint)

#Add three fields to the new layer, name of type ogr.OFTString and density and fraction of type ogr.OFTReal
field = ogr.FieldDefn("name", ogr.OFTString)
density_layer.CreateField(field)

field = ogr.FieldDefn("density", ogr.OFTReal)
density_layer.CreateField(field)

field = ogr.FieldDefn("fraction", ogr.OFTReal)
density_layer.CreateField(field)


#Look for the names ofs fields
locations_def = layer_wijken.GetLayerDefn()
num_fields = locations_def.GetFieldCount()
print("Task4: Number of fields ",num_fields)

#task 4: print the name and type of each field in the layer
for i in range(0,num_fields):
    print("Name of field:",locations_def.GetFieldDefn(i).GetName())
    print("Type of field", locations_def.GetFieldDefn(i).GetTypeName())

#Buurtcombinatie = districts?

#3. Iterate over the districts. For each district
#a. Get the name and the size of the area of the current district, and initialise variables to store the
#number and areas of houses
layer_def = layer_wijken.GetLayerDefn()
num_fields = layer_def.GetFieldCount()
num_features = layer_wijken.GetFeatureCount()

for i in range(1,num_features+1):
    feature = layer_wijken.GetFeature(i)
    name = feature.GetField('Buurtcombinatie')
    geometry = feature.GetGeometryRef()
    print(name, geometry.GetArea())

#b. Iterating over a layer works once. For a repeated iteration over a layer you need to use
#ResetReading() before you attempt to iterate another time:

#c. c. For each centroid test whether it is in the current district geometry. If so, accumulate the number and
#area. You can use Within to test the geometries:
density_layer_def = density_layer.GetLayerDefn()

for x in tqdm(range(1, layer_wijken.GetFeatureCount()+1)):  #for each feature in "wijken" (districts)
    district_feature = layer_wijken.GetFeature(x)
    district_geometry = district_feature.GetGeometryRef()
    centroid = district_geometry.Centroid()
    district_area = district_geometry.GetArea()
    name = district_feature["Buurtcombinatie"] #get the district name
    houses = 0
    area = 0
    layer_centroids.ResetReading()

    for y in range(1, layer_centroids.GetFeatureCount()+1):

        centroid_feature = layer_centroids.GetFeature(y)
        centroid_geometry = centroid_feature.GetGeometryRef()

        if centroid_geometry.Within(district_geometry):
            houses += 1
            area += centroid_feature.area

    #d. Compute the density and fraction and assign these with the name of the current district to the output #layer
    density = houses / (district_area / 1000000)
    fraction = area / district_area * 100
    point_feature = ogr.Feature(density_layer_def) # create a new feature
    point = ogr.Geometry(ogr.wkbPoint) # create a point geometry
    point.AddPoint(centroid.GetX(), centroid.GetY()) # set the coordinates of this point
    point_feature.SetGeometry(point)
    point_feature.SetField('name', name)
    point_feature.SetField('density', density)
    point_feature.SetField('fraction', fraction)

    density_layer.CreateFeature(point_feature)

## Question: What is the density of the district with feature id 54 (Museumkwartier)?
c_db = ogr.GetDriverByName('GPKG').Open("centroids.gpkg", update=0)
density = c_db.GetLayerByName('density')
density.GetFeature(54).GetField("density")
```

Get_school_data.py

```python
# -*- coding: utf-8 -*-
"""
Created on Fri Nov 20 17:09:46 2020

@author: hansf
"""

import requests

#query on dataset where po = primary schools
r = requests.get('https://schoolwijzer.amsterdam.nl/nl/api/v1/lijst/po/', verify = False)
print(r.json())

#save file to disk
import json
with open('data.json', 'w', encoding='utf-8') as f:
    json.dump(r.json(), f, ensure_ascii=False, indent=4)


```

Convert_json.py

```python
import json
from osgeo.osr import SpatialReference, CoordinateTransformation
from osgeo import ogr, gdal

#Assign the CRS to Amsterdam
rdNew = SpatialReference()
rdNew.ImportFromEPSG(28992)

#Create the dataset and add a layer with:
school_source = ogr.GetDriverByName('GPKG').Open('schools.gpkg', update=1)

if school_source.GetLayerByName('locations'):
    school_source.DeleteLayer('locations')
    print('Layer districts removed!!!')
point_layer = school_source.CreateLayer('locations', srs=rdNew, geom_type=ogr.wkbPoint)

# add a new layer buffer. The layer will be used to store the new features.
rdNew = SpatialReference()
rdNew.ImportFromEPSG(28992)

#check if buffer layer exists alreadry and remove it
if school_source.GetLayerByName('districts'):
    school_source.DeleteLayer('districts')
    print('Layer districts removed!!!')

#add new layer to the dataset
districts_layer = school_source.CreateLayer('districts', srs=rdNew, geom_type=ogr.wkbMultiPolygon)
districts_layer_def = districts_layer.GetLayerDefn()

#load data
with open('data.json') as json_file:
    school_data = json.load(json_file)

#set CRS
wgs84 = SpatialReference()
wgs84.ImportFromEPSG(4326)
wgs_to_rd = CoordinateTransformation(wgs84, rdNew)

point_layer_def = point_layer.GetLayerDefn()

#iterate over all schools and assign each school lat / long to a point
for s in school_data['results']:
    latitude=s['coordinaten']['lat']
    longitude=s['coordinaten']['lng']
    #eliminate 0,0 points
    if not(latitude==0 and longitude==0):
        point = wgs_to_rd.TransformPoint(latitude, longitude)
        rd_x=point[0]
        rd_y=point[1]
        feature = ogr.Feature(point_layer_def)
        point = ogr.Geometry(ogr.wkbPoint)
        point.AddPoint(rd_x, rd_y)
        feature.SetGeometry(point)
        point_layer.CreateFeature(feature)
```

Create_buffer.py

```python
# -*- coding: utf-8 -*-
"""
Created on Fri Nov 20 17:44:29 2020

@author: hansf
"""

import json
from osgeo.osr import SpatialReference, CoordinateTransformation
from osgeo import ogr, gdal

data_source = ogr.GetDriverByName('GPKG').Open('schools.gpkg', update=1)
point_layer = data_source.GetLayerByName('locations')
# add a new layer buffer. The layer will be used to store the new features.
rdNew = SpatialReference()
rdNew.ImportFromEPSG(28992)
#check if buffer layer exists alreadry and remove it
if data_source.GetLayerByName('buffer'):
    data_source.DeleteLayer('buffer')

    print('Layer buffer removed!!!')
#add new layer to the dataset
buffer_layer = data_source.CreateLayer('buffer', srs=rdNew, geom_type=ogr.wkbPolygon)
buffer_layer_def = buffer_layer.GetLayerDefn()
buffer_distance=250
for c in range(1,point_layer.GetFeatureCount()+1):
    point_feature=point_layer.GetFeature(c)
    point_geometry = point_feature.GetGeometryRef()
    buffer_geometry = point_geometry.Buffer(buffer_distance)
    #create new feature
    feature = ogr.Feature(buffer_layer_def)
    #set new feature's geometry
    feature.SetGeometry(buffer_geometry)
    #add new feature to the layer
    buffer_layer.CreateFeature(feature)
```

Merge_buffer.py

```python
import json
from osgeo.osr import SpatialReference, CoordinateTransformation
from osgeo import ogr, gdal

#Assign the CRS to Amsterdam
rdNew = SpatialReference()
rdNew.ImportFromEPSG(28992)

data_source = ogr.GetDriverByName('GPKG').Open('schools.gpkg', update=1)
buffer_layer = data_source.GetLayerByName('buffer')

#create a new layer buffer to store new features
if data_source.GetLayerByName("merge"):
    data_source.DeleteLayer("merge")
merge_layer = data_source.CreateLayer('merge', srs=rdNew, geom_type=ogr.wkbMultiPolygon)
merge_layer_def = buffer_layer.GetLayerDefn()

#Afterwards create a new feature merge_feature and add the geometry of the first buffer feature to the new feature.
#Also add a geometry merge_geometry and initialise it with the geometry of your first feature. You will use this
#geometry to construct the merged buffer area.
buffer_feature = buffer_layer.GetNextFeature()
buffer_geometry = buffer_feature.GetGeometryRef()
#create new feature
merge_feature = ogr.Feature(merge_layer_def)
#add geometry of buffer feature to new merge feature
merge_feature.SetGeometry(buffer_geometry)
merge_geometry= merge_feature.GetGeometryRef()

for c in range(1,buffer_layer.GetFeatureCount()+1):
    # Get the geometry of the current buffer feature
    buffer_feature=buffer_layer.GetFeature(c)
    buffer_geometry = buffer_feature.GetGeometryRef()
    # Merge the current buffer geometry with the previously merged area
    union = merge_geometry.Union(buffer_geometry)
    # Create a feature with the merged geometry
    merge_feature = ogr.Feature(merge_layer_def)
    merge_feature.SetGeometry(union)
    # update merge_geometry
    merge_geometry= merge_feature.GetGeometryRef()
#create new feature
feature = ogr.Feature(merge_layer_def)
#set new feature's geometry
feature.SetGeometry(merge_geometry)
#add new feature to the layer
merge_layer.CreateFeature(feature)
```

Merge_discrit.py (school_away is in the same code!)

```python
import json
from osgeo.osr import SpatialReference, CoordinateTransformation
from osgeo import ogr, gdal

#Assign the CRS to Amsterdam
rdNew = SpatialReference()
rdNew.ImportFromEPSG(28992)

data_source = ogr.GetDriverByName('GPKG').Open('schools.gpkg', update=1)

wijken_data_source = ogr.GetDriverByName('GPKG').Open("Amsterdam_BAG.gpkg", update=1)
wijken_layer = wijken_data_source.GetLayerByName('Wijken')

rdNew = SpatialReference()
rdNew.ImportFromEPSG(28992)

if data_source.GetLayerByName("districts"):
    data_source.DeleteLayer("districts")
districts_layer = data_source.CreateLayer('districts', srs=rdNew, geom_type=ogr.wkbMultiPolygon)
districts_feature_def = districts.GetLayerDefn() # define new layer

wijken_feature = wijken_layer.GetNextFeature() # get first feature of buffer
wijken_geometry = wijken_feature.GetGeometryRef()

districts_feature = ogr.Feature(districts_feature_def) # initialize the merge_feature
districts_feature.SetGeometry(wijken_geometry)
districts_geometry = districts_feature.GetGeometryRef() # set the merge_geometry

for i in range(1,len(wijken_layer)):
    wijken_feature = wijken_layer.GetNextFeature()
    wijken_geometry = wijken_feature.GetGeometryRef()

    union = districts_geometry.Union(wijken_geometry)
    districts_feature.SetGeometry(union)
    districts_geometry = districts_feature.GetGeometryRef()

# Save the new feature to the the districs layer
districts_layer.CreateFeature(districts_feature)

#school_away code
merge_layer = data_source.GetLayerByName('merge')
merge_layer_def = merge_layer.GetLayerDefn()


# and add a new layer away
if data_source.GetLayerByName('away'):
    data_source.DeleteLayer("away")
away = data_source.CreateLayer('away', srs=rdNew, geom_type=ogr.wkbMultiPolygon)
away_feature_def = away.GetLayerDefn() # define new layer
away_feature = ogr.Feature(away_feature_def)

districts_layer.SymDifference(merge_layer, away)
feature_away = away.GetFeature(1)
gem_away = feature_away.GetGeometryRef()
print('The area considered far away from schools is (m2): ', gem_away.GetArea())
```