Assignment 2.2 – Geometric Manipulations

Due date: 23th November 2020

Name: Lena Walcher

Student ID: 2818833

1.1 Study area

In this assignment about geometric manipulation the dataset from The Dutch Basisregistratie Adressen en Gebouwden “BAG” was used to analyze different layers and data in the city of Amsterdam. The figure below shows the districts of Amsterdam in grey and the surface areas of the building in purple.

2. Querying datasets

The python script explore_bag.py was created in order to get a good overview over the data.

To print the number of layers included in the dataset the commands

```
print("Task1: ",data_source.GetLayerCount())
print("End TASK \n")
```

were used, showing an output of 3 layers for the dataset.

To print the CRS of each layer the commands

```
# print the CRS for the layers
layer = data_source.GetLayerByIndex()
srs = layer.GetSpatialRef()
print("Task2: Used CRS \n\n", srs)
print("End Task2 \n")
```

which showed that "Amersfoort / RD New" was used for this dataset.

In order of showing the layer name and CRS of each layer, as well as the number of features included in each layer, the following lines of code were used

```
# print feature count of each layer
buildings = data_source.GetLayerByName('Verblijfsobject')
number_ft_buildings = buildings.GetFeatureCount()
print("Feature count of Verblijfsobject:\n", number_ft_buildings)
surface_area = data_source.GetLayerByName('Pand')
number_ft_area = surface_area.GetFeatureCount()
print("Feature count of Pand:\n", number_ft_area)
districts = data_source.GetLayerByName('Wijken')
number_ft_districts = districts.GetFeatureCount()
print("Feature count of Wijken:\n", number_ft_districts )
print("End Task3 \n")
locations_def= buildings.GetLayerDefn()
surface_def= surface_area.GetLayerDefn()
districts_def =districts.GetLayerDefn()
# buildings
print("Definition of Layer buildings:\n", buildings.GetLayerDefn())
# number of fields in the layer
print("Number of fields in buildings:\n", locations_def.GetFieldCount())
# surface_area
print("Definition of Layer surface_area:\n", surface_area.GetLayerDefn())
# number of fields in the layer
print("Number of fields in surface_areas:\n\n", surface_def.GetFieldCount())
# districts
print("Definition of Layer districts:\n", districts.GetLayerDefn())
# number of fields in the layer
print("Number of fields in district:\n\n", districts_def.GetFieldCount())
# just layer Verblijfsobject
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())
```

The result showed that the feature count of Verblijfsobject is 496420, the feature count of Pand is 182747 and the feature count of Wijken is 99. There is two fields in the layer “buildings”, one in the layer “surface_areas” and one field in the layer “districts”.

The field in the buildings layer is called gebruiksdoel and of type string. The layer district contains the fields oppervlakte and gebruiksdoel of type integer and string.

```
#task 4: print the name and type of each field in the layer
num_fields = locations_def.GetFieldCount()
print("Task4: Number of fields ",num_fields, "\n")
```

To iterate over all features in the layer with the goal of adding up the whole surface area of oppervlakte (buildings) to a resulting total of 59255192, the following lines of code were used.

```
# task 5: value of field "oppervlakte" (surface area) with all features in the layer added up
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, "\n\n")
```

Question 1: What is the total surface area given in the location layer?

The total surface area in the location layer is 59255192.

Question 2: What is the coordinate of the feature with the index 439774?

The feature with the index 439774 is of point geometry with the coordinates of 121815.991 for its x-coordinates and 487912.647 for its y-coordinates.

2.2 Obtaining building properties

In order of calculating the surface area of each building a new Python script "building_surface_areas.py" was created. The aim was to find out whether a building belongs to a certain district one has to calculate each buildings centroid. In order to achieve this, a new GeoPackage was created, as week as a new layer to store mentioned centroids in.

```
# create a new layer to store the centroids
# assign a CRS to the new layer - Amersfoort / RD New (28992).
rdNew = SpatialReference()
rdNew.ImportFromEPSG(28992)
# new dataset with an output layer centroids  with point geometry type
centroid_source = ogr.GetDriverByName('GPKG').CreateDataSource('centroids.gpkg')
centroid_layer = centroid_source.CreateLayer('centroids', srs=rdNew, geom_type=ogr.wkbPoint)
# Task: Add field area the layer centroids.
field = ogr.FieldDefn('area', ogr.OFTReal)
centroid_layer.CreateField(field)
# feature definition
centroid_layer_def = centroid_layer.GetLayerDefn()
```

In the next step the field "area" was added to the layer "centroids" to then calculate the area of the location of each centroid to get one closer to the goal of finding out what area of Amsterdam a specific building belongs to.

```
# Task: Calculate the area and the centroid location of each building.
# using a loop to iterate over each feature
for i in range(1, surface.GetFeatureCount()):
    feature = surface.GetFeature(i)
    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() # apply centroid
    point.AddPoint(centroid.GetX(), centroid.GetY()) #  point geometry for feature
    point_feature.SetGeometry(point) # place of the centroid to the new point
    house_area = house_geometry.GetArea()
    # set the value of a field and add feature to the layer
    point_feature.SetField('area', house_area)
    centroid_layer.CreateFeature(point_feature)
```

As one can see in the visualization to the left, yellow points were added to the green building surfaces representing the centroids of each building. One should notice too, that the yellow dots are not always within the boundaries or geometry of a building as the centroid can be outside of the building.

2.3 Computing building densities

The aim of the following part was to create a density map of all houses in Amsterdam by using the centroid layer that was created earlier. In order to do so, a new output layer “density” was created and the fields name, density and fraction were added.

```
# create new output layer for density in the centroids dataset
# use the same CRS
rdNew = SpatialReference()
rdNew.ImportFromEPSG(28992)
# check to see of density layer already exists and remove it
if centroid_source.GetLayerByName('density'):
    centroid_source.DeleteLayer('density')
    print('Layer density removed!\n')
#add new layer to the dataset
```

```
density_layer = centroid_source.CreateLayer('density', srs=rdNew, geom_type=ogr.wkbPoint)
# add fields to the density layer
# name
field_name = ogr.FieldDefn('name', ogr.OFTString)
density_layer.CreateField(field_name)
# density
field_density = ogr.FieldDefn('density', ogr.OFTReal)
density_layer.CreateField(field_density)
# fraction
field_fraction = ogr.FieldDefn('fraction', ogr.OFTReal)
density_layer.CreateField(field_fraction)
#Look for the names of fields
locations_def = layer_wijken.GetLayerDefn()
num_fields = locations_def.GetFieldCount()
print("Task4: The 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())
```

In order to get name and size of the district area it was necessary to store initialize variables to store the houses in.

```
#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:
#   centroid_layer.ResetReading()
# C. density_layer_def = density_layer.GetLayerDefn()
 for x in tqdm(range(1, layer_wijken.GetFeatureCount()+1)):  #for each feature in the districts layer
    district_feature = layer_wijken.GetFeature(x) #get the specific feature of layer
    district_geometry = district_feature.GetGeometryRef() #get the geometry of the feature
    centroid = district_geometry.Centroid() # get the centroid of the geometry (feature)
    district_area = district_geometry.GetArea() #get the area of the geometry
    name = district_feature["Buurtcombinatie"] #get the district name
```

```
    houses = 0
    area = 0
    centroids.ResetReading()
    for y in range(1, centroids.GetFeatureCount()+1):  #for each feature in centroids layer
        centroid_feature = 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)
#d. Compute the density and fraction and assign these with the name of the current district to the output layer
## 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")
```

Question 3: What is the density of the district with feature id 54 (Museumkwartier)?

The density of the district Museumkwartier is 1385402.37.

3. Working with Amsterdam School Data

3.1 Retrieving school data

For the next step the script get-school_data.py was created to initially retrieve data about Amsterdams schools. The data was then stored in a JSON file called schools.json.

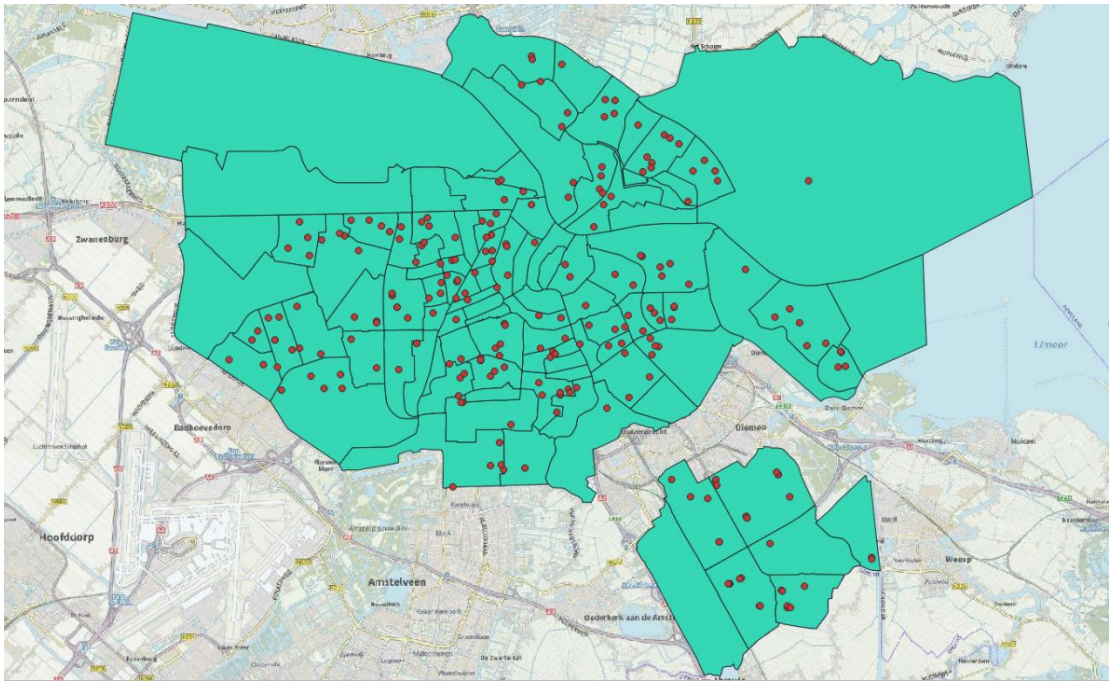
```
with open('schools.json', 'w') as f:
    json.dump(schools, f)
f.close()
# Assign CRS
rdNew = SpatialReference()
rdNew.ImportFromEPSG(28992)
rdWSG84 = SpatialReference()
rdWSG84.ImportFromEPSG(4326)
# Create dataset and add layer
ogr_ds = ogr.GetDriverByName('GPKG').CreateDataSource('schools.gpkg')
point_layer = ogr_ds.CreateLayer('locations', srs=rdNew, geom_type=ogr.wkbPoint)
with open('schools.json') as f:
    school_data = json.load(f)
```

The data from the created json file was then used to create the variable “school_data” wich was then used to extract the coordinates of each school within that variable.

```
# Task: Complete your Python script to create the school layer and run it.
wgs_to_rd = CoordinateTransformation(rdWSG84, rdNew) # create transformation for the reference systems
# For better orientation name, id and bring were added to the layer:
field1 = ogr.FieldDefn('name', ogr.OFTString)
field2 = ogr.FieldDefn('id', ogr.OFTInteger)
field3 = ogr.FieldDefn('brin', ogr.OFTString)
# Add the fields to the layer:
point_layer.CreateField(field1)
point_layer.CreateField(field2)
point_layer.CreateField(field3)
#defenition for the layer
feature_def = point_layer.GetLayerDefn()
for i in school_data.get('results'): # loop through all schools
    coordinaten = i.get('coordinaten') # dict.
    latitude = coordinaten.get('lat') # latitude
    longitude = coordinaten.get('lng') # longitude
    if latitude != 0 and longitude != 0:  # only coordinates not zero:
        schoolpoint = wgs_to_rd.TransformPoint(latitude, longitude) # transform RS
        rd_x = schoolpoint[0]  # get the x and y an name it
        rd_y = schoolpoint[1]
```

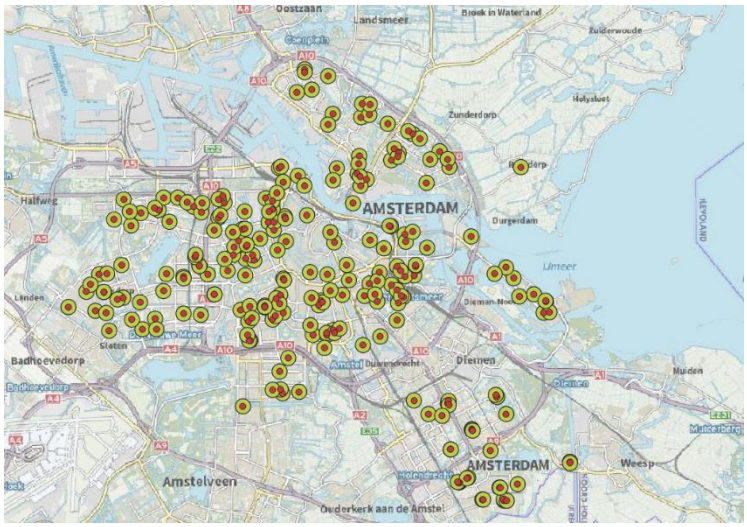
```
feature = ogr.Feature(feature_def) # initialize feature
point = ogr.Geometry(ogr.wkbPoint) # initialize point
point.AddPoint(rd_x, rd_y) # set values to point
feature.SetGeometry(point) # add point to feature
name = i.get('naam') # name school
school_id = i.get('id') # id school
brin = i.get('brin') # brin school
# add values of fields to feature:
feature.SetField('name', name)
feature.SetField('id', school_id)
feature.SetField('brin', brin)
point_layer.CreateFeature(feature)
```

With this code the school layer was created. As seen in the visual below once can distinct the locations of all schools as dots within the districts of Amsterdam.



3.3 Adding buffer areas around schools

In the following section we added a 250m buffer around each school to determine an area that would be considered “close” to each school.



When visualizing this step as shown on the left, one can see the green circle around each school (red point) representing the buffer of 250 meters.

In order to achieve this visual, the new layer “buffer” was created to store the new features. All features in the point layer had to be iterated in order to retrieve the point geometry.

```
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 was 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)
```

Question 4: What is the geometry type of the layer buffer?

The geometry type of the layer buffer is stored as a multipolygon.

3.4 Merging geometries

To compute the area that is considered far away from schools and therefore outside of a 250m reach to the next school, two new layers had to be created. The first layer “merge” contains the merged school buffers and the second feature with the merge layer the merged districts “merge_feature”.

The script “merge_buffer.py” contains all steps.

```
# and add a new layer merge
rdNew = SpatialReference()
rdNew.ImportFromEPSG(28992)
merge = data_source.CreateLayer('merge', srs=rdNew, geom_type=ogr.wkbMultiPolygon)
merge_feature_def = merge.GetLayerDefn() # define new layer
# Add a new feature, merge_feature and initialize it with the geometry of the first
# feature of the buffer layer. After that, iterate over the rest of the buffer layers
# to merge the rest of the buffer layer.
buffer_feature = buffer_layer.GetNextFeature() # get first feature of buffer
buffer_geometry = buffer_feature.GetGeometryRef()
merge_feature = ogr.Feature(merge_feature_def) # initialize the merge_feature
merge_feature.SetGeometry(buffer_geometry)
merge_geometry = merge_feature.GetGeometryRef() # set the merge_geometry
for i in range(1,len(buffer_layer)):
    buffer_feature = buffer_layer.GetNextFeature()
    buffer_geometry = buffer_feature.GetGeometryRef()
    union = merge_geometry.Union(buffer_geometry)
    merge_feature.SetGeometry(union)
    merge_geometry = merge_feature.GetGeometryRef()
# Save the new feature to the the merge layer
merge.CreateFeature(merge_feature)
```

Question 5: What is the geometry type of the layer merge?

The layer merge is of type multipolygon.

3.5 Compute area far from schools

In the final step of this assignment the area far away from public schools is calculated in the merge_districts.py script, mentioned earlier. First the new feature was added to the district layer and then the new layer “away” was added. The layer “away” was created and then the OGR operation Erase was applied, in order to receive the entire layer to exclude the areas that are considered close to schools (250m radius) around each school within Amsterdam.

```
# new feature added to the existing layer
districts.CreateFeature(districts_feature)
merge_layer = data_source.GetLayerByName('merge')
merge_layer_def = merge_layer.GetLayerDefn()
```

```
districts_layer = districts
# add layer "away" to the merge layer
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 "far away" from schools in m2 is: ', gem_away.GetArea())
```

As a result one can see the buffer areas that are shown in the left visualization above being erased out of the visual on the right, leaving the area of Amsterdam that is considered far away from public schools and therefore outside of a 250 radius to the next school.

Question 6: Which operation will you use to compute the area far away from schools?

a) Clip
b) Intersection
X c) Erase
d) Difference

Question 7: What is the size of the area considered as far away from public schools?

a) 0.14 km2
b) 168.32 km2
c) 186.64 km2
d) 192.16 km2

The area that is considered far away from schools is 186648064.65494493m2, which rounds to 168.85 square kilometers.

Attachments:

1. explore_bag
2. building_surface_areas
3. densities
4. get_school_data
5. create_buffer
6. merge_buffer
7. merge_districts

Attachment 1

```
# -*- coding: utf-8 -*-
"""
Created on Thu Nov 19 14:32:58 2020

@author: lenaw
"""
# 2.1

from osgeo import gdal, ogr

# loading the data
filename = 'Amsterdam_BAG.gpkg'
data_source = ogr.GetDriverByName('GPKG').Open(filename, update=0)

#Task1: Print the number of Layers Included in Dataset:
# get number of layers with
print("Task1: ",data_source.GetLayerCount())
print("End TASK \n")

# print the CRS for the layers
layer = data_source.GetLayerByIndex()
srs = layer.GetSpatialRef()
print("Task2: Used CRS \n\n", srs)
print("End Task2 \n")

# print feature count of each layer
buildings = data_source.GetLayerByName('Verblijfsobject')
number_ft_buildings = buildings.GetFeatureCount()
print("Feature count of Verblijfsobject:\n", number_ft_buildings)

surface_area = data_source.GetLayerByName('Pand')
number_ft_area = surface_area.GetFeatureCount()
print("Feature count of Pand:\n", number_ft_area)

districts = data_source.GetLayerByName('Wijken')
number_ft_districts = districts.GetFeatureCount()
print("Feature count of Wijken:\n", number_ft_districts )
print("End Task3 \n")


locations_def= buildings.GetLayerDefn()
surface_def= surface_area.GetLayerDefn()
districts_def =districts.GetLayerDefn()

# buildings
print("Definition of Layer buildings:\n", buildings.GetLayerDefn())
# number of fields in the layer
print("Number of fields in buildings:\n", locations_def.GetFieldCount())

# surface_area
print("Definition of Layer surface_area:\n", surface_area.GetLayerDefn())
# number of fields in the layer
```

```
print("Number of fields in surface_areas:\n\n", surface_def.GetFieldCount())

# districts
print("Definition of Layer districts:\n", districts.GetLayerDefn())
# number of fields in the layer
print("Number of fields in district:\n\n", districts_def.GetFieldCount())

# Names and Type of each layer
# buildings
print("Name of field: \n", locations_def.GetFieldDefn(1).GetName())
print("Type of field:\n", locations_def.GetFieldDefn(1).GetTypeName())
print("End Task4 \n")

#task 4: print the name and type of each field in the layer
num_fields = locations_def.GetFieldCount()
print("Task4: Number of fields ",num_fields, "\n")

# just layer Verblijfsobject
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())


# task 5: value of field "oppervlakte" (surface area) with all features in the layer added up
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, "\n\n")
# Question: What is the total surface area given in the location layer?
# Anwer: The total surface area in the location layer is "Total area: 59255192"

# query individual features with index through geometry
feature = buildings.GetFeature(439774)
geometry = feature.GetGeometryRef()
print(geometry)
print("X coordinate", geometry.GetX())
print("Y coordinate", geometry.GetY())

# Question: What is the coordinate of the feature with the index 439774
# Answer: POINT (121815.991 487912.647 0)
# X 121815.991
# Y 487912.647
```

Attachment 2

```
# -*- coding: utf-8 -*-
"""
Created on Thu Nov 19 18:31:36 2020

@author: lenaw
"""
from osgeo import gdal, ogr
from osgeo.osr import SpatialReference

data_source = ogr.GetDriverByName('GPKG').Open("Amsterdam_BAG.gpkg", update=0)
surface = data_source.GetLayerByName('Pand')

# create a new layer to store the centroids
# assign a CRS to the new layer - Amersfoort / RD New (28992).
rdNew = SpatialReference()
rdNew.ImportFromEPSG(28992)

# Create the new dataset with an output layer centroids ny using the point geometry type:
centroid_source = ogr.GetDriverByName('GPKG').CreateDataSource('centroids.gpkg')
centroid_layer = centroid_source.CreateLayer('centroids', srs=rdNew, geom_type=ogr.wkbPoint)

# Task: Add field area the layer centroids.
field = ogr.FieldDefn('area', ogr.OFTReal)
centroid_layer.CreateField(field)

# first need to get feature definition
centroid_layer_def = centroid_layer.GetLayerDefn()


# Task: Calculate the area and the centroid location of each building.
# using a loop to iterate over each feature
for i in range(1, surface.GetFeatureCount()):
    feature = surface.GetFeature(i)
    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() # apply centroid
    point.AddPoint(centroid.GetX(), centroid.GetY()) #  point geometry for feature
    point_feature.SetGeometry(point) # place of the centroid to the new point

    house_area = house_geometry.GetArea()
    # set the value of a field and add feature to the layer
    point_feature.SetField('area', house_area)
    centroid_layer.CreateFeature(point_feature)
```

Attachment 3

```
# -*- coding: utf-8 -*-
"""
Created on Fri Nov 20 17:41:38 2020

@author: lenaw
"""

from osgeo import gdal, ogr
from osgeo.osr import SpatialReference
from tqdm import tqdm

# load the sources
data_source = ogr.GetDriverByName('GPKG').Open("Amsterdam_BAG.gpkg", update=0)
centroid_source = ogr.GetDriverByName('GPKG').Open('centroids.gpkg', update=1)


# open the surface layer Wijken
layer_wijken = data_source.GetLayerByName('Wijken')
# open the centroids layer from centroid_source
centroids = centroid_source.GetLayerByName('centroids')
# open the density layer from centroid_source
density =  centroid_source.GetLayerByName('density')


# create new output layer for density in the centroids dataset
# use the same CRS
rdNew = SpatialReference()
rdNew.ImportFromEPSG(28992)

# check to see of density layer already exists and remove it
if centroid_source.GetLayerByName('density'):
    centroid_source.DeleteLayer('density')
    print('Layer density removed!\n')

#add new layer to the dataset
density_layer = centroid_source.CreateLayer('density', srs=rdNew, geom_type=ogr.wkbPoint)

# add fields to the density layer
# name
field_name = ogr.FieldDefn('name', ogr.OFTString)
density_layer.CreateField(field_name)

# density
field_density = ogr.FieldDefn('density', ogr.OFTReal)
density_layer.CreateField(field_density)

# fraction
field_fraction = ogr.FieldDefn('fraction', ogr.OFTReal)
density_layer.CreateField(field_fraction)


#Look for the names ofs fields
```

```
locations_def = layer_wijken.GetLayerDefn()
num_fields = locations_def.GetFieldCount()
print("Task4: The 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())


#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:
#   centroid_layer.ResetReading()

#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 the districts layer
    district_feature = layer_wijken.GetFeature(x) #get the specific feature of layer
    district_geometry = district_feature.GetGeometryRef() #get the geometry of the feature
    centroid = district_geometry.Centroid() # get the centroid of the geometry (feature)
    district_area = district_geometry.GetArea() #get the area of the geometry
    name = district_feature["Buurtcombinatie"] #get the district name

    houses = 0
    area = 0

    centroids.ResetReading()

    for y in range(1, centroids.GetFeatureCount()+1):  #for each feature in centroids layer

        centroid_feature = 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)

#d. Compute the density and fraction and assign these with the name of the current district to the output
#layer

## 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")
```

Attachment 4

```
# -*- coding: utf-8 -*-
"""
Created on Fri Nov 20 21:00:04 2020

@author: lenaw
"""
#use python request module to perform request in Schoolwijyer API
import requests
import json
from osgeo.osr import SpatialReference, CoordinateTransformation
from osgeo import ogr, gdal

# import the data
schools = requests.get('https://schoolwijzer.amsterdam.nl/api/v1/lijst/po', verify=False)
schools = schools.json()

# Task: Create a Python script get_school_data.py that executes an API request to the OpenData API,
# and writes the result to a file named schools.json.
# Use Python's json module to write the file to disk.

with open('schools.json', 'w') as f:
    json.dump(schools, f)
f.close()

# Assign CRS
rdNew = SpatialReference()
rdNew.ImportFromEPSG(28992)

rdWSG84 = SpatialReference()
rdWSG84.ImportFromEPSG(4326)

# Create dataset and add layer
ogr_ds = ogr.GetDriverByName('GPKG').CreateDataSource('schools.gpkg')
point_layer = ogr_ds.CreateLayer('locations', srs=rdNew, geom_type=ogr.wkbPoint)

with open('schools.json') as f:
    school_data = json.load(f)

print(type(school_data))

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

wgs_to_rd = CoordinateTransformation(rdWSG84, rdNew) # create transformation for the reference systems

# For better orientation name, id and bring were added to the layer:
```

```
field1 = ogr.FieldDefn('name', ogr.OFTString)
field2 = ogr.FieldDefn('id', ogr.OFTInteger)
field3 = ogr.FieldDefn('brin', ogr.OFTString)

# Add the fields to the layer:
point_layer.CreateField(field1)
point_layer.CreateField(field2)
point_layer.CreateField(field3)

#defenition for the layer
feature_def = point_layer.GetLayerDefn()

for i in school_data.get('results'): # loop through all schools
    coordinaten = i.get('coordinaten') # dict.
    latitude = coordinaten.get('lat') # latitude
    longitude = coordinaten.get('lng') # longitude


    if latitude != 0 and longitude != 0:  # only coordinates not zero:
        schoolpoint = wgs_to_rd.TransformPoint(latitude, longitude) # transform RS
        rd_x = schoolpoint[0]  # get the x and y an name it
        rd_y = schoolpoint[1]

        feature = ogr.Feature(feature_def) # initialize feature
        point = ogr.Geometry(ogr.wkbPoint) # initialize point
        point.AddPoint(rd_x, rd_y) # set values to point
        feature.SetGeometry(point) # add point to feature

        name = i.get('naam') # name school
        school_id = i.get('id') # id school
        brin = i.get('brin') # brin school

        # add values of fields to feature:
        feature.SetField('name', name)
        feature.SetField('id', school_id)
        feature.SetField('brin', brin)
        point_layer.CreateFeature(feature)
```

Attachment 5

```
1.  # -*- coding: utf-8 -*-
2.  """
3.  Created on Sat Nov 21 10:08:20 2020
4.
5.  @author: lenaw
6.  """
7.
8.  import json
9.  from osgeo.osr import SpatialReference, CoordinateTransformation
10. from osgeo import ogr, gdal
11.
12. data_source = ogr.GetDriverByName('GPKG').Open('schools.gpkg', update=1)
13. point_layer = data_source.GetLayerByName('locations')
14.
15. # add a new layer buffer. The layer will be used to store the new features.
16. rdNew = SpatialReference()
17. rdNew.ImportFromEPSG(28992)
18.
19. #check if buffer layer exists alreadry and remove it
20. if data_source.GetLayerByName('buffer'):
21.     data_source.DeleteLayer('buffer')
22.     print('Layer buffer was removed!')
23.
24. #add new layer to the dataset
25. buffer_layer = data_source.CreateLayer('buffer', srs=rdNew, geom_type=ogr.wkbPolygon)
26. buffer_layer_def = buffer_layer.GetLayerDefn()
27. buffer_distance=250
28. for c in range(1,point_layer.GetFeatureCount()+1):
29.     point_feature=point_layer.GetFeature(c)
30.     point_geometry = point_feature.GetGeometryRef()
31.     buffer_geometry = point_geometry.Buffer(buffer_distance)
32.     #create new feature
33.     feature = ogr.Feature(buffer_layer_def)
34.     #set new feature's geometry
35.     feature.SetGeometry(buffer_geometry) #add new feature to the layer
36.     buffer_layer.CreateFeature(feature)
```

Attachment 6

```
# -*- coding: utf-8 -*-
"""
Created on Sat Nov 21 13:55:00 2020

@author: lenaw
"""

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

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

# and add a new layer merge
rdNew = SpatialReference()
rdNew.ImportFromEPSG(28992)

merge = data_source.CreateLayer('merge', srs=rdNew, geom_type=ogr.wkbMultiPolygon)
merge_feature_def = merge.GetLayerDefn() # define new layer
# Add a new feature, merge_feature and initialize it with the geometry of the first
# feature of the buffer layer. After that, iterate over the rest of the buffer layers
# to merge the rest of the buffer layer.

buffer_feature = buffer_layer.GetNextFeature() # get first feature of buffer
buffer_geometry = buffer_feature.GetGeometryRef()

merge_feature = ogr.Feature(merge_feature_def) # initialize the merge_feature
merge_feature.SetGeometry(buffer_geometry)
merge_geometry = merge_feature.GetGeometryRef() # set the merge_geometry

for i in range(1,len(buffer_layer)):
    buffer_feature = buffer_layer.GetNextFeature()
    buffer_geometry = buffer_feature.GetGeometryRef()

    union = merge_geometry.Union(buffer_geometry)
    merge_feature.SetGeometry(union)
    merge_geometry = merge_feature.GetGeometryRef()

# Save the new feature to the the merge layer
merge.CreateFeature(merge_feature)
```

Attachment 7

```
# -*- coding: utf-8 -*-
"""
Created on Sat Nov 21 13:59:28 2020

@author: lenaw
"""
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')

# and add a new layer merge
rdNew = SpatialReference()
rdNew.ImportFromEPSG(28992)

if data_source.GetLayerByName("districts"):
    data_source.DeleteLayer("districts")
districts = 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)
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()

```

```
# new feature added to the existing layer
districts.CreateFeature(districts_feature)

merge_layer = data_source.GetLayerByName('merge')
merge_layer_def = merge_layer.GetLayerDefn()

districts_layer = districts

# add layer "away" to the merge layer
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 "far away" from schools in m2 is: ', gem_away.GetArea())
```