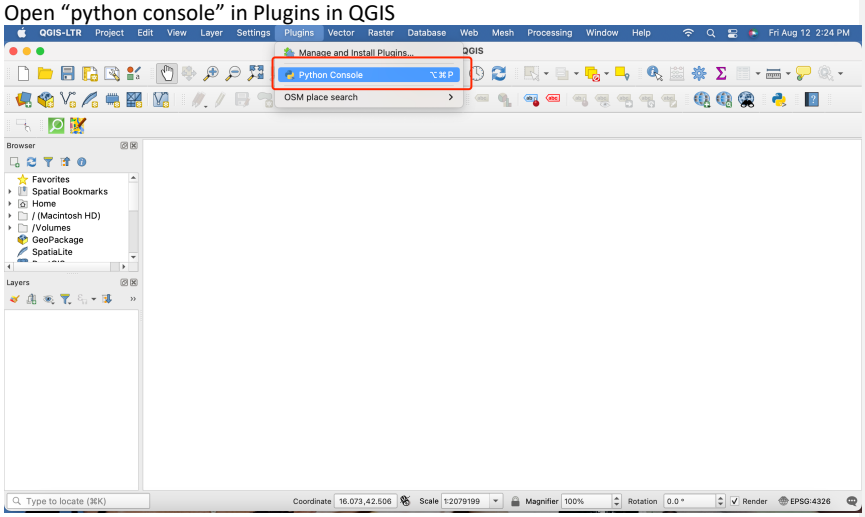
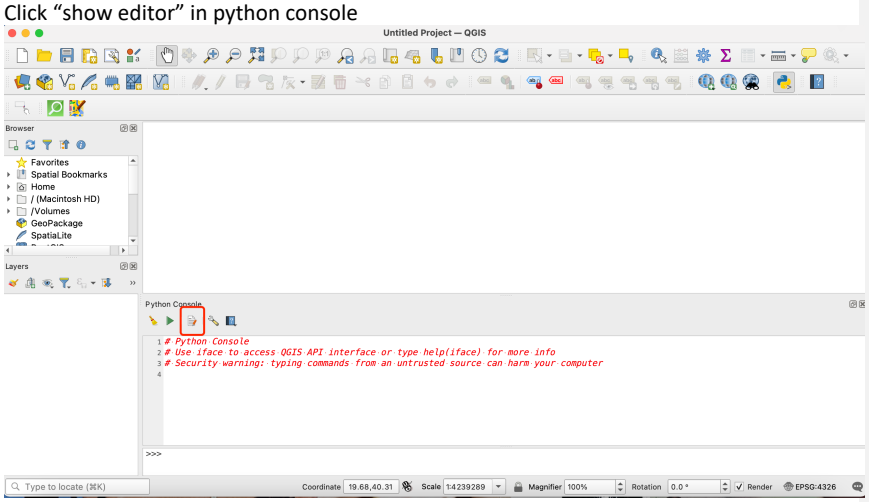
Steps for using this code template:

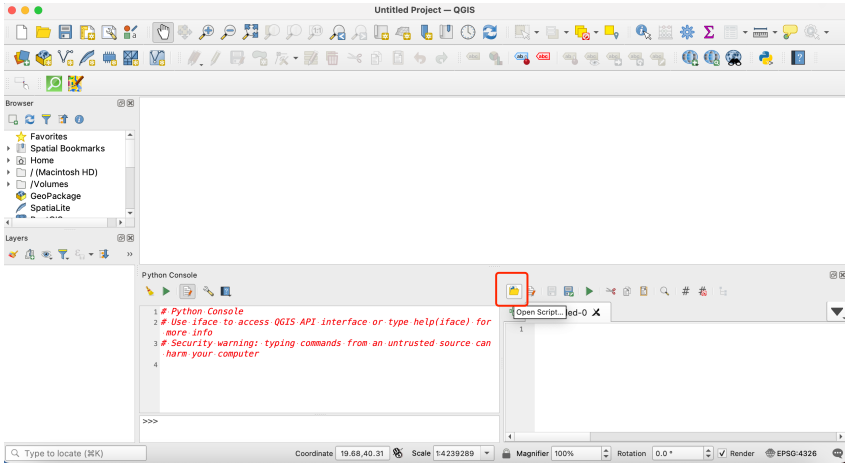
1. Open “python console” in Plugins in QGIS



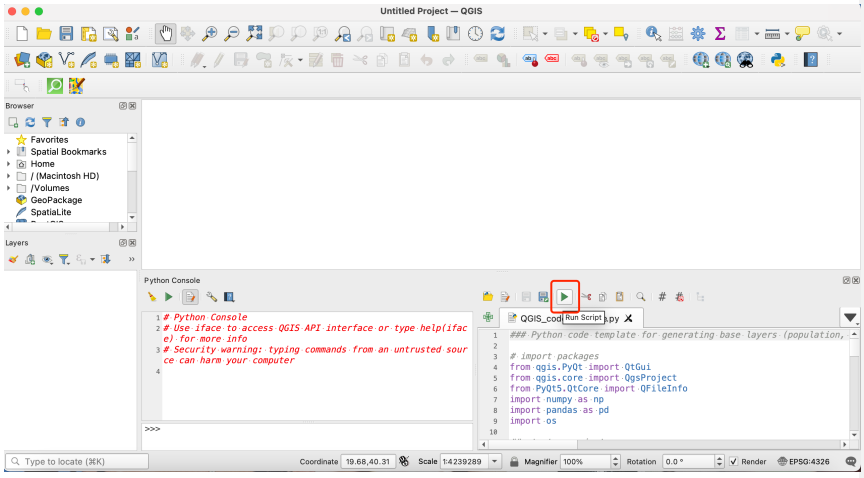
2. Click “show editor” in python console



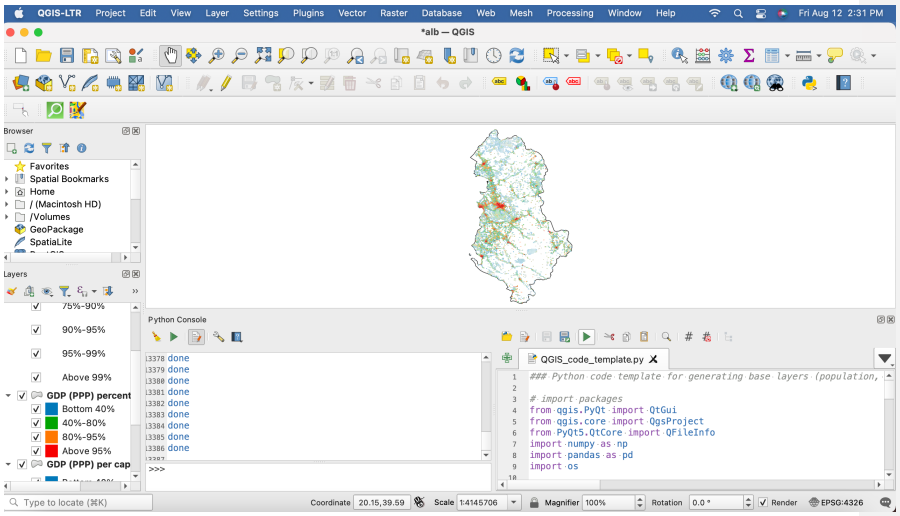
3. Click “open script” in editor and read in the code template



4. Click “run script”’
   Note: this will take some time, please wait patiently until layers show up



5. The final result will look like this:



Code template break-down:

```
### Python code template for generating base layers (population, GDP PPP, GDP PC) in
QGIS

# import packages
from qgis.PyQt import QtGui
from qgis.core import QgsProject
from PyQt5.QtCore import QFileInfo
import numpy as np
import pandas as pd
import os

## start a project
project = QgsProject.instance()
project_path =
"/Users/yangshining/Desktop/internnnnn/IFC_summer_intern/IFC_intern_Shining/mapping_pr
oject/IDH project/Pyqgis code scripts/alb.qgz"


## load vector layer – admin boundary of the country
uri1 =
"/Users/yangshining/Desktop/internnnnn/IFC_summer_intern/IFC_intern_Shining/mapping_pr
oject/Albania/2 Raw Data/Administrative Boundaries/alb_admbnda_adm0_2019c.shp"
```

Commented [ys1]: Path where this map project will be stored

Commented [ys2]: Path of the country admin boundary shapefile

```
vec_layer = QgsVectorLayer(uri1, "countrys", "ogr")  # layer name and provider name
single_symbol_renderer = vec_layer.renderer()
symbol = single_symbol_renderer.symbol()
# Set fill colour
symbol.setColor(QColor.fromRgb(255, 255, 255))
QgsProject.instance().addMapLayer(vec_layer)
# Refresh
vec_layer.triggerRepaint()
iface.layerTreeView().refreshLayerSymbology(vec_layer.id())


## add csv file
# set file name
filename = "Albania_final_data_2022_0706_1km.csv"
# set file path
filepath =
"/Users/yangshining/Desktop/internnnnn/IFC_summer_intern/IFC_intern_Shining/mapping_pr
oject/Albania/3 Output Data/"
# combine file path and name, prepare for uri
combine = "file:///" + filepath + filename
# build uri
uri2 = combine + "?encoding=%s&delimiter=%s&xField=%s&yField=%s&crs=%s" % (
    "UTF-8",
    ",",
    "longitud",
    "lat",
    "epsg:4326",
)
csv_layer = QgsVectorLayer(uri2, "final_data", "delimitedtext")
if not vec_layer.isValid():
    print("Layer not loaded")

QgsProject.instance().addMapLayer(csv_layer)
# set the points invisible
QgsProject.instance().layerTreeRoot().findLayer(csv_layer).setItemVisibilityChecked(
    False
)

## create square buffer around output points

inputfile = "final_data"
outputfile =
"/Users/yangshining/Desktop/internnnnn/IFC_summer_intern/IFC_intern_Shining/mapping_pr
oject/IDH_project/Pyqgis_code_scripts/bufferzone.shp"
bufferDist = 0.00415
Data = QgsProject.instance().mapLayersByName(inputfile)
layer = Data[0]
fields = layer.fields()
```

Commented [ys3]: Layer name – could be country name

Commented [ys4]: The file name of the output csv file

Commented [ys5]: Folder where the output csv data is stored g

Commented [ys6]: If run on Windows, drop a "/" here

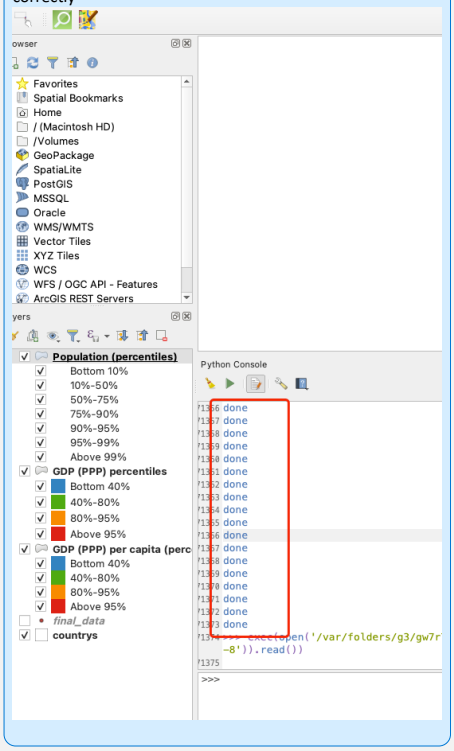
Commented [ys7]: Make sure these names are compatible with what in the csv file – some could be "Long" and "Lat"

Commented [ys8]: Path where the bufferzone shapefile is stored

Commented [ys9]: Buffer distance

```
writer = QgsVectorFileWriter(
    outputfile,
    "UTF-8",
    fields,
    QgsWkbTypes.Polygon,
    layer.sourceCrs(),
    "ESRI Shapefile",
)
for f in layer.getFeatures():
    geom = f.geometry()
    buffer = geom.buffer(
        bufferDist, 1, QgsGeometry.CapSquare, QgsGeometry.JoinStyleMiter, 2.0
    )
    f.setGeometry(buffer)
    writer.addFeature(f)
    print("done")

# iface.addVectorLayer(outputfile, "GDP (PPP) per capita (percentiles)", "ogr")
del writer
# iface.addVectorLayer(outputfile, "GDP (PPP) percentiles", "ogr")
# iface.addVectorLayer(outputfile, "Population (percentiles)", "ogr")


## generate graduated renderer based on percentiles

# read in output csv file
df = pd.read_csv(os.path.join(filepath, filename))
# get population cutoffs
pop_values = df["population"]
pop_cutoff = np.nanquantile(pop_values, [0.1, 0.5, 0.75, 0.9, 0.95, 0.99, 1])
# get GDP (PPP) cutoffs
gdp_values = df["GDP_PPP"]
gdp_cutoff = np.nanquantile(gdp_values, [0.4, 0.8, 0.95, 1])
# get GDP (PPP) per capita cutoffs
gdp_pc_values = df["pc"]
gdp_pc_cutoff = np.nanquantile(gdp_pc_values, [0.4, 0.8, 0.95, 1])

# create layer - GDP (PPP) per capita
GDP_PC = QgsVectorLayer(outputfile, "GDP (PPP) per capita (percentiles)", "ogr")
targetField = "pc"  # column name
rangeList = []
opacity = 1

# define 1st value ranges
minVal = 0
maxVal = gdp_pc_cutoff[0]
# range label
lab1 = "Bottom 40%"
```

Commented [ys10]: Note: Python console printing out a series of “done” means that the buffering part is running correctly



```
# color (blue)
rangeColor = QtGui.QColor.fromRgb(31, 120, 180)
# create symbol and set properties
symbol1 = QgsSymbol.defaultSymbol(GDP_PC.geometryType())
symbol1.setColor(rangeColor)
symbol1.setOpacity(opacity)
symbol1[0].setStrokeColor(QColor("Transparent"))
symbol1.symbolLayer(0).setStrokeStyle(Qt.PenStyle(Qt.SolidLine))
# create range and append to rangeList
range1 = QgsRendererRange(minVal, maxVal, symbol1, lab1)
rangeList.append(range1)

# define 2nd value ranges
minVal = gdp_pc_cutoff[0]
maxVal = gdp_pc_cutoff[1]
# range label
lab2 = "40%-80%"
# color (green)
rangeColor = QtGui.QColor.fromRgb(51, 160, 40)
# create symbol and set properties
symbol2 = QgsSymbol.defaultSymbol(GDP_PC.geometryType())
symbol2.setColor(rangeColor)
symbol2.setOpacity(opacity)
symbol2[0].setStrokeColor(QColor("Transparent"))
symbol2.symbolLayer(0).setStrokeStyle(Qt.PenStyle(Qt.SolidLine))
# create range and append to rangeList
range2 = QgsRendererRange(minVal, maxVal, symbol2, lab2)
rangeList.append(range2)

# define 3rd value ranges
minVal = gdp_pc_cutoff[1]
maxVal = gdp_pc_cutoff[2]
# range label
lab3 = "80%-95%"
# color (orange)
rangeColor = QtGui.QColor.fromRgb(255, 127, 0)
# create symbol and set properties
symbol3 = QgsSymbol.defaultSymbol(GDP_PC.geometryType())
symbol3.setColor(rangeColor)
symbol3.setOpacity(opacity)
symbol3[0].setStrokeColor(QColor("Transparent"))
symbol3.symbolLayer(0).setStrokeStyle(Qt.PenStyle(Qt.SolidLine))
# create range and append to rangeList
range3 = QgsRendererRange(minVal, maxVal, symbol3, lab3)
rangeList.append(range3)

# define 4th value ranges
minVal = gdp_pc_cutoff[2]
```

```
maxVal = gdp_pc_cutoff[3]
# range label
lab4 = "Above 95%"
# color (red)
rangeColor = QtGui.QColor.fromRgb(227, 26, 28)
# create symbol and set properties
symbol4 = QgsSymbol.defaultSymbol(GDP_PC.geometryType())
symbol4.setColor(rangeColor)
symbol4.setOpacity(opacity)
symbol4[0].setStrokeColor(QColor("Transparent"))
symbol4.symbolLayer(0).setStrokeStyle(Qt.PenStyle(Qt.SolidLine))
# create range and append to rangeList
range4 = QgsRendererRange(minVal, maxVal, symbol4, lab4)
rangeList.append(range4)
# create the renderer
groupRenderer = QgsGraduatedSymbolRenderer("", rangeList)
groupRenderer.setMode(QgsGraduatedSymbolRenderer.EqualInterval)
groupRenderer.setClassAttribute(targetField)
# apply renderer to layer
GDP_PC.setRenderer(groupRenderer)
# add to QGIS interface
QgsProject.instance().addMapLayer(GDP_PC)


## create layer - GDP (PPP)
GDP_PPP = QgsVectorLayer(outputfile, "GDP (PPP) percentiles", "ogr")
targetField = "GDP_PPP"  # column name
rangeList = []
opacity = 1

# define 1st value ranges
minVal = 0
maxVal = gdp_cutoff[0]
# range label
lab1 = "Bottom 40%"
# color (blue)
rangeColor = QtGui.QColor.fromRgb(31, 120, 180)
# create symbol and set properties
symbol1 = QgsSymbol.defaultSymbol(GDP_PPP.geometryType())
symbol1.setColor(rangeColor)
symbol1.setOpacity(opacity)
symbol1[0].setStrokeColor(QColor("Transparent"))
symbol1.symbolLayer(0).setStrokeStyle(Qt.PenStyle(Qt.SolidLine))
# create range and append to rangeList
range1 = QgsRendererRange(minVal, maxVal, symbol1, lab1)
rangeList.append(range1)

# define 2nd value ranges
```

```
minVal = gdp_cutoff[0]
maxVal = gdp_cutoff[1]
# range label
lab2 = "40%–80%"
# color (green)
rangeColor = QtGui.QColor.fromRgb(51, 160, 40)
# create symbol and set properties
symbol2 = QgsSymbol.defaultSymbol(GDP_PPP.geometryType())
symbol2.setColor(rangeColor)
symbol2.setOpacity(opacity)
symbol2[0].setStrokeColor(QColor("Transparent"))
symbol2.symbolLayer(0).setStrokeStyle(Qt.PenStyle(Qt.SolidLine))
# create range and append to rangeList
range2 = QgsRendererRange(minVal, maxVal, symbol2, lab2)
rangeList.append(range2)

# define 3rd value ranges
minVal = gdp_cutoff[1]
maxVal = gdp_cutoff[2]
# range label
lab3 = "80%–95%"
# color (orange)
rangeColor = QtGui.QColor.fromRgb(255, 127, 0)
# create symbol and set properties
symbol3 = QgsSymbol.defaultSymbol(GDP_PPP.geometryType())
symbol3.setColor(rangeColor)
symbol3.setOpacity(opacity)
symbol3[0].setStrokeColor(QColor("Transparent"))
symbol3.symbolLayer(0).setStrokeStyle(Qt.PenStyle(Qt.SolidLine))
# create range and append to rangeList
range3 = QgsRendererRange(minVal, maxVal, symbol3, lab3)
rangeList.append(range3)

# define 4th value ranges
minVal = gdp_cutoff[2]
maxVal = gdp_cutoff[3]
# range label
lab4 = "Above 95%"
# color (red)
rangeColor = QtGui.QColor.fromRgb(227, 26, 28)
# create symbol and set properties
symbol4 = QgsSymbol.defaultSymbol(GDP_PPP.geometryType())
symbol4.setColor(rangeColor)
symbol4.setOpacity(opacity)
symbol4[0].setStrokeColor(QColor("Transparent"))
symbol4.symbolLayer(0).setStrokeStyle(Qt.PenStyle(Qt.SolidLine))
# create range and append to rangeList
range4 = QgsRendererRange(minVal, maxVal, symbol4, lab4)
```

```
rangeList.append(range4)
# create the renderer
groupRenderer = QgsGraduatedSymbolRenderer("", rangeList)
groupRenderer.setMode(QgsGraduatedSymbolRenderer.EqualInterval)
groupRenderer.setClassAttribute(targetField)
# apply renderer to layer
GDP_PPP.setRenderer(groupRenderer)
# add to QGIS interface
QgsProject.instance().addMapLayer(GDP_PPP)


## create layer - population
pop = QgsVectorLayer(outputfile, "Population (percentiles)", "ogr")
targetField = "population"  # column name
rangeList = []
# range color (white)
rangeColor = QtGui.QColor("white")

# define 1st value ranges
opacity1 = 1
minVal = 0
maxVal = pop_cutoff[0]
# range label
lab1 = "Bottom 10%"
# create symbol and set properties
symbol1 = QgsSymbol.defaultSymbol(pop.geometryType())
symbol1.setColor(rangeColor)
symbol1.setOpacity(opacity1)
symbol1[0].setStrokeColor(QColor("Transparent"))
# create range and append to rangeList
range1 = QgsRendererRange(minVal, maxVal, symbol1, lab1)
rangeList.append(range1)

# define 2nd value ranges
opacity2 = 1 - (np.log(pop_cutoff[0] + 1) / np.log(pop_cutoff[5] + 1))
minVal = pop_cutoff[0]
maxVal = pop_cutoff[1]
# range label
lab2 = "10%-50%"
# create symbol and set properties
symbol2 = QgsSymbol.defaultSymbol(pop.geometryType())
symbol2.setColor(rangeColor)
symbol2.setOpacity(opacity2)
symbol2[0].setStrokeColor(QColor("Transparent"))
# create range and append to rangeList
range2 = QgsRendererRange(minVal, maxVal, symbol2, lab2)
rangeList.append(range2)
```

```
# define 3rd value ranges
opacity3 = 1 - (np.log(pop_cutoff[1] + 1) / np.log(pop_cutoff[5] + 1))
minVal = pop_cutoff[1]
maxVal = pop_cutoff[2]
# range label
lab3 = "50%-75%"
# create symbol and set properties
symbol3 = QgsSymbol.defaultSymbol(pop.geometryType())
symbol3.setColor(rangeColor)
symbol3.setOpacity(opacity3)
symbol3[0].setStrokeColor(QColor("Transparent"))
# create range and append to rangeList
range3 = QgsRendererRange(minVal, maxVal, symbol3, lab3)
rangeList.append(range3)

# define 4th value ranges
opacity4 = 1 - (np.log(pop_cutoff[2] + 1) / np.log(pop_cutoff[5] + 1))
minVal = pop_cutoff[2]
maxVal = pop_cutoff[3]
# range label
lab4 = "75%-90%"
# create symbol and set properties
symbol4 = QgsSymbol.defaultSymbol(pop.geometryType())
symbol4.setColor(rangeColor)
symbol4.setOpacity(opacity4)
symbol4[0].setStrokeColor(QColor("Transparent"))
# create range and append to rangeList
range4 = QgsRendererRange(minVal, maxVal, symbol4, lab4)
rangeList.append(range4)

# define 5th value ranges
opacity5 = 1 - (np.log(pop_cutoff[3] + 1) / np.log(pop_cutoff[5] + 1))
minVal = pop_cutoff[3]
maxVal = pop_cutoff[4]
# range label
lab5 = "90%-95%"
# create symbol and set properties
symbol5 = QgsSymbol.defaultSymbol(pop.geometryType())
symbol5.setColor(rangeColor)
symbol5.setOpacity(opacity5)
symbol5[0].setStrokeColor(QColor("Transparent"))
# create range and append to rangeList
range5 = QgsRendererRange(minVal, maxVal, symbol5, lab5)
rangeList.append(range5)

# define 6th value ranges
opacity6 = 1 - (np.log(pop_cutoff[4] + 1) / np.log(pop_cutoff[5] + 1))
minVal = pop_cutoff[4]
```

```
maxVal = pop_cutoff[5]
# range label
lab6 = "95%–99%"
# create symbol and set properties
symbol6 = QgsSymbol.defaultSymbol(pop.geometryType())
symbol6.setColor(rangeColor)
symbol6.setOpacity(opacity6)
symbol6[0].setStrokeColor(QColor("Transparent"))
# create range and append to rangeList
range6 = QgsRendererRange(minVal, maxVal, symbol6, lab6)
rangeList.append(range6)

# define 7th value ranges
opacity7 = 0
minVal = pop_cutoff[5]
maxVal = pop_cutoff[6]
# range label
lab7 = "Above 99%"
# create symbol and set properties
symbol7 = QgsSymbol.defaultSymbol(pop.geometryType())
symbol7.setColor(rangeColor)
symbol7.setOpacity(opacity7)
symbol7[0].setStrokeColor(QColor("Transparent"))
# create range and append to rangeList
range7 = QgsRendererRange(minVal, maxVal, symbol7, lab7)
rangeList.append(range7)
# create the renderer
groupRenderer = QgsGraduatedSymbolRenderer("", rangeList)
groupRenderer.setMode(QgsGraduatedSymbolRenderer.EqualInterval)
groupRenderer.setClassAttribute(targetField)
# apply renderer to layer
pop.setRenderer(groupRenderer)
# add to QGIS interface
QgsProject.instance().addMapLayer(pop)

## save the layers to geopackage
output_path =
'/Users/yangshining/Desktop/internnnnn/IFC_summer_intern/IFC_intern_Shining/mapping_pr
oject/IDH_project/Pyqgis_code_scripts/'
pop_path = output_path + "pop.gpkg"
GDP_PPP_path = output_path + "GDP_PPP.gpkg"
GDP_PC_path = output_path + "GDP_PC.gpkg"
# layers = [styled_layer]
processing.run(
    "native:package",
    {"LAYERS": pop, "OUTPUT": pop_path, "OVERWRITE": False, "SAVE_STYLES": True},
)
processing.run(
```

Commented [ys11]: Folder where the layer geopackages will be stored

```
    "native:package",
    {
        "LAYERS": GDP_PPP,
        "OUTPUT": GDP_PPP_path,
        "OVERWRITE": False,
        "SAVE_STYLES": True,
    },
)
processing.run(
    "native:package",
    {"LAYERS": GDP_PC, "OUTPUT": GDP_PC_path, "OVERWRITE": False, "SAVE_STYLES":
True},
)


## add health facilities - healthsite.io
# set file name
health_io = "Albania_healthsites_io.csv"
# set file path
filepath =
'/Users/yangshining/Desktop/internnnnn/IFC_summer_intern/IFC_intern_Shining/mapping_pr
oject/Albania/3_Output_Data/"
# combine file path and name, prepare for uri
combine = "file:///" + filepath + health_io
# build uri
uri3 = combine + "?encoding=%s&delimiter=%s&xField=%s&yField=%s&crs=%s" % (
    "UTF-8",
    ",",
    "Long",
    "Lat",
    "epsg:4326",
)
health_io_layer = QgsVectorLayer(uri3, "Healthsite.io", "delimitedtext")
if not health_io_layer.isValid():
    print("Layer not loaded")
QgsProject.instance().addMapLayer(health_io_layer)


## add health facilities - data from WHO
# set file name
health_WHO = "Albania_WHO_health_facilities.csv"
# set file path
filepath =
"/Users/yangshining/Desktop/internnnnn/IFC_summer_intern/IFC_intern_Shining/mapping_pr
oject/Albania/3_Output_Data/"
# combine file path and name, prepare for uri
combine = "file:///" + filepath + health_WHO
# build uri
```

Commented [ys12]: File name of the healthsite.io csv

Commented [ys13]: Output data folder

Commented [ys14]: If run on Windows, drop a "/" here

Commented [ys15]: Note: some countries might not have WHO data, and we can just delete these WHO-related parts in the code

Commented [ys16]: File name of the WHO health facilities csv

Commented [ys17]: Output data folder

Commented [ys18]: If run on Windows, drop a "/" here

```
uri4 = combine + "?encoding=%s&delimiter=%s&xField=%s&yField=%s&crs=%s" % (
    "UTF-8",
    ",",
    "Long",
    "Lat",
    "epsg:4326",
)
health_WHO_layer = QgsVectorLayer(uri4, "Health facilities - WHO", "delimitedtext")
if not health_WHO_layer.isValid():
    print("Layer not loaded")
QgsProject.instance().addMapLayer(health_WHO_layer)


## Move the health facility data points to the top
# health facilities - WHO
alayer = QgsProject.instance().mapLayersByName("Health facilities - WHO")[0]
root = QgsProject.instance().layerTreeRoot()
myalayer = root.findLayer(alayer.id())
myClone = myalayer.clone()
parent = myalayer.parent()
parent.insertChildNode(0, myClone)
parent.removeChildNode(myalayer)

# healthsite.io
alayer = QgsProject.instance().mapLayersByName("Healthsite.io")[0]
root = QgsProject.instance().layerTreeRoot()
# Move Layer
myalayer = root.findLayer(alayer.id())
myClone = myalayer.clone()
parent = myalayer.parent()
parent.insertChildNode(0, myClone)
parent.removeChildNode(myalayer)


## save the project
project.write(project_path)
```