**Modularity of Task**

For the task of analyzing the geospatial data in a QGIS environment, let's break it down into modules:

1. Module for Loading Data: Functions to load vector and raster layers.

2. Module for Processing Geometry: Functions to calculate centroids and geometrical properties of features.

3. Module for Raster Analysis: Functions to extract raster values at specific points.

4. Module for Creating Output Layer: Functions to create a new vector layer and populate it with the processed data.

5. Module for Saving Data: Functions to save the new vector layer to a file.

# Import QGIS Python modules

from qgis.core import (

QgsVectorLayer, QgsRasterLayer, QgsProject, QgsFeature, QgsField, QgsPointXY, QgsVectorFileWriter

)

from qgis.PyQt.QtCore import QVariant

import processing

# Module for Loading Data

def load\_vector\_layer(path, name):

layer = QgsVectorLayer(path, name, "ogr")

if not layer.isValid():

raise Exception(f"Failed to load vector layer: {name}")

QgsProject.instance().addMapLayer(layer)

return layer

def load\_raster\_layer(path, name):

layer = QgsRasterLayer(path, name)

if not layer.isValid():

raise Exception(f"Failed to load raster layer: {name}")

QgsProject.instance().addMapLayer(layer)

return layer

# Module for Processing Geometry

def calculate\_centroid(feature):

return feature.geometry().centroid()

# Module for Raster Analysis

def get\_raster\_value\_at\_point(raster\_layer, point):

(result, values) = raster\_layer.dataProvider().sample(QgsPointXY(point), 1)

if result:

return values[0]

return None

# Module for Creating Output Layer

def create\_centroid\_layer(fields):

centroid\_layer = QgsVectorLayer('Point?crs=epsg:4326', 'Boulder Centroids', 'memory')

pr = centroid\_layer.dataProvider()

pr.addAttributes(fields)

centroid\_layer.updateFields()

return centroid\_layer

# Module for Saving Data

def save\_layer\_to\_file(layer, output\_path, driver\_name="ESRI Shapefile"):

options = QgsVectorFileWriter.SaveVectorOptions()

options.driverName = driver\_name

QgsVectorFileWriter.writeAsVectorFormat(layer, output\_path, options)

# Main Function

def main(vector\_path, raster\_path, output\_path):

try:

vector\_layer = load\_vector\_layer(vector\_path, "Boulder Polygons")

raster\_layer = load\_raster\_layer(raster\_path, "Encoded Depths")

fields = [

QgsField("Poly\_ID", QVariant.String),

QgsField("Easting", QVariant.Double),

QgsField("Northing", QVariant.Double),

QgsField("WaterDepth", QVariant.Double),

# Add other fields as needed

]

centroid\_layer = create\_centroid\_layer(fields)

for feature in vector\_layer.getFeatures():

centroid = calculate\_centroid(feature)

depth = get\_raster\_value\_at\_point(raster\_layer, centroid.asPoint())

new\_feature = QgsFeature()

new\_feature.setGeometry(centroid)

new\_feature.setAttributes([feature["Poly\_ID"], centroid.asPoint().x(), centroid.asPoint().y(), depth])

centroid\_layer.dataProvider().addFeature(new\_feature)

save\_layer\_to\_file(centroid\_layer, output\_path)

print("Process completed successfully.")

except Exception as e:

print(f"An error occurred: {e}")

# Replace with your file paths and run the main function

vector\_file\_path = 'path\_to\_your\_shapefile/Boulder polygons.shp'

raster\_file\_path = 'path\_to\_your\_raster/Test\_Encoded\_Depths\_File.tif'

output\_file\_path = 'path\_to\_your\_output\_directory/Boulder\_centroids.shp'

main(vector\_file\_path, raster\_file\_path, output\_file\_path)

Each module has a specific responsibility, and they can be used independently or replaced as needed without affecting the rest of the program. This modular script is more maintainable, and parts of it can be reused in other projects with similar requirements.