

PROOVITÖÖ ÜLESANNE

1 Eesmärk

Proovitöö ülesandeks on saada arvulised vastused ülesande detailses kirjelduses (punkt 2) toodud küsimustele. Iga ülesande puhul on nõutud detailne lahenduskäigu kirjeldus, vastus küsimusele ja ülesande teostamise kuupäev. Kuupäev on vajalik Tellijale proovitöö vastuste kontrollimisel, kuna andmed võivad olla muutumises.

2 Ülesande detailsem kirjeldus

2.1 Kui palju (mitu tk) on WFS-teenuse andmetel 2023 aastal inventeeritud metsaeraldisi, kus peapuuliigiks on olnud tamm?

2.2 Kui palju (mitu tk) on punktis 2.1 leitud eraldisi inventeeritud 2023.aastal Lääne maakonnas (kattuvuse päring keskpunkti järgi)?

2.3 Mis on punktis 2.1 leitud eraldiste pindala (hektarites) kokku?

2.4 Mitu kehtivat metsateatist on punktis 2.1 leitud eraldistel?

3 Proovitööks vajalikud andmed

3.1 Proovitöö ülesande lahendamiseks tuleb kasutada Kliimaministeeriumi haldusalas kasutusel oleva avaliku Geoserveri kaardikihte aadressil <https://gsavalik.envir.ee/geoserver/> .

3.2 Maakondade piiride ruumikujud leiab 'ehak' nimeruumis asuvast kihist [ehak:maakondade_piirid](#).

3.3 Ülejäänud andmed vastuste tarvis tuleb leida avaliku Geoserveri 'metsaregister' nimeruumis asuvatest kihtidest.

WFS Server Capabilities and Configuration

The **GetCapabilities** request returns information about the capabilities and configuration of the WFS server.

<https://gsavalik.envir.ee/geoserver/wfs?request=GetCapabilities&service=WFS>

The response will be an XML containing settings and available layers. Available layers can be found in the **FeatureTypeList** element.

```
334 </ows:AllowedValues>
335 </ows:Constraint>
336 </ows:OperationsMetadata>
337 <FeatureTypeList>
338 <FeatureType xmlns:vmk="https://gsavalik.envir.ee/vmk">
339 <Name>vmk:vmk_meetm_koik</Name>
340 <Title># Veemajanduskavade kõik meetmed</Title>
341 <Abstract>Sisaldab kõiki kaardikihte, mis on ka ühekaupa teenuses välja toodud. Andmete allikaks on Kliimaministeerium või Keskkonnaag
342 <DefaultCRS>urn:ogc:def:crs:EPSG::3301</DefaultCRS>
343 <OtherCRS>urn:ogc:def:crs:EPSG::3857</OtherCRS>
344 <OtherCRS>urn:ogc:def:crs:EPSG::4326</OtherCRS>
345 <ows:WGS84BoundingBox>
346 <ows:LowerCorner>20.100786540955877 57.44973890996587</ows:LowerCorner>
347 <ows:UpperCorner>28.50902437846779 60.06548397644703</ows:UpperCorner>
348 </ows:WGS84BoundingBox>
349 <MetadataURL xlink:href="https://metadata.geoportaal.ee/geonetwork/srv/eng/catalog.search#/metadata/bfcd3ffa-31bb-4742-8eea-8ef5b84393
350 <MetadataURL xlink:href="https://metadata.geoportaal.ee/geonetwork/srv/api/records/bfcd3ffa-31bb-4742-8eea-8ef5b84393fa/formatters/xml
351 </FeatureType>
352 <FeatureType xmlns:kponahtused="kpois_kponahtus"...>
353 <FeatureType xmlns:kmanahtused="kpois_kmanahtus">
354 <Name>kmanahtused:kma_avalik_gaas_351</Name>
355 <Title>A ja B kategooria gaasipaigaldis - gaasipaigaldise kaitsevöönd - gaas</Title>
356 <Abstract>Andmed pärinevad kitsenduste (KPOIS) andmebaasist. Andmeid haldab Maa-ameti katastriosakond. Geomeetriaveeru nimi on 'shape'
357 <ows:Keywords>
358 <ows:Keyword>gaas</ows:Keyword>
359 <ows:Keyword>gaasipaigaldis</ows:Keyword>
360 <ows:Keyword>kitsendused</ows:Keyword>
361 <ows:Keyword>Maa-amet</ows:Keyword>
362 <ows:Keyword>mõjuala</ows:Keyword>
363 <ows:Keyword>KPOIS</ows:Keyword>
364 <ows:Keyword>A kategooria</ows:Keyword>
```

The layers and spatial data sets referenced in the test assignment are detailed in the response.

Layer: ehak:maakondade_piirid

```
10744 </FeatureType>
10745 <FeatureType xmlns:ehak="kemit.ee/ehak">
10746   <Name>ehak:maakondade_piirid</Name>
10747   <Title>Maakondade piirid</Title>
10748   <Abstract>Eesti maakondade piiride ruumikujud kehtiva seisuga, uuendatakse iga ööpäev. Andmeid haldab Maa-amet.</Abstract>
10749   <ows:Keywords>
10750     <ows:Keyword>ehak</ows:Keyword>
10751     <ows:Keyword>kataster</ows:Keyword>
10752     <ows:Keyword>maa-amet</ows:Keyword>
10753     <ows:Keyword>haldusüksus</ows:Keyword>
10754     <ows:Keyword>haldusjaotus</ows:Keyword>
10755     <ows:Keyword>maakond</ows:Keyword>
10756     <ows:Keyword>maakonnad</ows:Keyword>
10757     <ows:Keyword>maakonna piir</ows:Keyword>
10758     <ows:Keyword>haldusüksuse piir</ows:Keyword>
10759     <ows:Keyword>eesti maakonnad</ows:Keyword>
10760     <ows:Keyword>WMS</ows:Keyword>
10761     <ows:Keyword>WFS</ows:Keyword>
10762     <ows:Keyword>kehtiv</ows:Keyword>
10763   </ows:Keywords>
10764   <DefaultCRS>urn:ogc:def:crs:EPSG::3301</DefaultCRS>
10765   <OtherCRS>urn:ogc:def:crs:EPSG::3857</OtherCRS>
10766   <OtherCRS>urn:ogc:def:crs:EPSG::4326</OtherCRS>
10767   <ows:WGS84BoundingBox>
10768     <ows:LowerCorner>20.100786540955884 57.449738940424304</ows:LowerCorner>
10769     <ows:UpperCorner>28.509024378467785 60.065484002788196</ows:UpperCorner>
10770   </ows:WGS84BoundingBox>
10771   <MetadataURL xlink:href="https://metadata.geoportaal.ee/geonetwork/srv/api/records/maaamet_haldus_asustus/formatters/xml"/>
10772   <MetadataURL xlink:href="https://metadata.geoportaal.ee/geonetwork/srv/api/records/maaamet_haldus_asustus"/>
10773 </FeatureType>
10774 <FeatureType xmlns:kmanhtused="kpois_kmanhtus">
```

Layers of the Spatial Data Set metsaregister:

```
12009 <ows:WGS84BoundingBox>
12010   <ows:LowerCorner>20.100786540955884 57.449738940424304</ows:LowerCorner>
12011   <ows:UpperCorner>28.509024378467785 60.065484002788196</ows:UpperCorner>
12012 </ows:WGS84BoundingBox>
12013 </FeatureType>
12014 <FeatureType xmlns:etak="ee.maaamet.etak">
12035 <FeatureType xmlns:eelis="http://kemit.ee/eelis/avaandmed">
12051 <FeatureType xmlns:metsaregister="https://mets-ave.envir.ee">
12067 <FeatureType xmlns:piiratud="http://kemit.ee/piiratud">
12083 <FeatureType xmlns:metsaregister="https://mets-ave.envir.ee">
12099 <FeatureType xmlns:metsaregister="https://mets-ave.envir.ee">
12111 <FeatureType xmlns:metsaregister="https://mets-ave.envir.ee">
12124 <FeatureType xmlns:eelis="http://kemit.ee/eelis/avaandmed">
12140 <FeatureType xmlns:kitsendus_suunatud="https://kemit.ee/kitsendus_suunatud">
12162 <FeatureType xmlns:kpokitsendused="kpois_kpo">
```

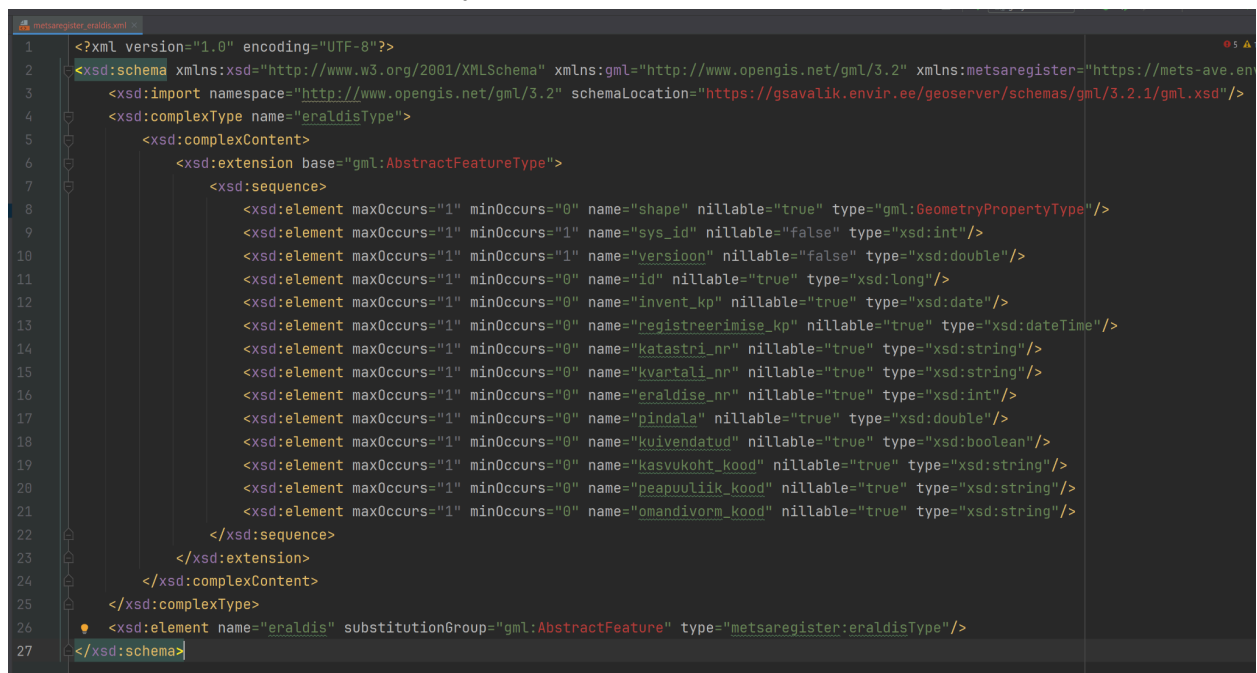
Task 2.1

For our work, we need the layer with forest plots **metsaregister:eraldis**. First, we will get information about this layer without geodata to understand what data it contains.

The layer description is obtained using the DescribeFeatureType command. The request will look like this:

<https://gsavalik.envir.ee/geoserver/wfs?service=WFS&version=2.0.0&request=DescribeFeatureType&typeName=metsaregister:eraldis>

The response is an XML with the layer description:



```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:gml="http://www.opengis.net/gml/3.2" xmlns:metsaregister="https://mets-ave.ee/geoserver/schemas/gml/3.2.1/gml.xsd" >
3   <xsd:import namespace="http://www.opengis.net/gml/3.2" schemaLocation="https://gsavalik.envir.ee/geoserver/schemas/gml/3.2.1/gml.xsd"/>
4   <xsd:complexType name="eraldisType">
5     <xsd:complexContent>
6       <xsd:extension base="gml:AbstractFeatureType">
7         <xsd:sequence>
8           <xsd:element maxOccurs="1" minOccurs="0" name="shape" nillable="true" type="gml:GeometryPropertyType"/>
9           <xsd:element maxOccurs="1" minOccurs="1" name="sys_id" nillable="false" type="xsd:int"/>
10          <xsd:element maxOccurs="1" minOccurs="1" name="version" nillable="false" type="xsd:double"/>
11          <xsd:element maxOccurs="1" minOccurs="0" name="id" nillable="true" type="xsd:long"/>
12          <xsd:element maxOccurs="1" minOccurs="0" name="invent_kp" nillable="true" type="xsd:date"/>
13          <xsd:element maxOccurs="1" minOccurs="0" name="registreerimise_kp" nillable="true" type="xsd:dateTime"/>
14          <xsd:element maxOccurs="1" minOccurs="0" name="katastri_nr" nillable="true" type="xsd:string"/>
15          <xsd:element maxOccurs="1" minOccurs="0" name="kvartali_nr" nillable="true" type="xsd:string"/>
16          <xsd:element maxOccurs="1" minOccurs="0" name="eraldis_nr" nillable="true" type="xsd:int"/>
17          <xsd:element maxOccurs="1" minOccurs="0" name="pindala" nillable="true" type="xsd:double"/>
18          <xsd:element maxOccurs="1" minOccurs="0" name="kuivendatud" nillable="true" type="xsd:boolean"/>
19          <xsd:element maxOccurs="1" minOccurs="0" name="kasvukoht_kood" nillable="true" type="xsd:string"/>
20          <xsd:element maxOccurs="1" minOccurs="0" name="peapuuliik_kood" nillable="true" type="xsd:string"/>
21          <xsd:element maxOccurs="1" minOccurs="0" name="omandivorm_kood" nillable="true" type="xsd:string"/>
22        </xsd:sequence>
23      </xsd:extension>
24    </xsd:complexContent>
25  </xsd:complexType>
26  <xsd:element name="eraldis" substitutionGroup="gml:AbstractFeature" type="metsaregister:eraldisType"/>
27 </xsd:schema>
```

According to the assignment, we are interested in the fields **invent_kp** and **peapuuliik_kood**. The inventory date is straightforward, but for the tree species **peapuuliik_kood**, we need additional information about possible values.

Tree Species Data

We search the server configuration file for information about tree species and find the layer with classifiers for species - **metsaregister:kl_puuliik**:

```

<FeatureType xmlns:metsaregister="https://mets-ave.envir.ee">
  <Name>metsaregister:kl_puuliik</Name>
  <Title>Klassifikaator: Puuliigid</Title>
  <Abstract/>
  <DefaultCRS>urn:ogc:def:crs:EPSG::404000</DefaultCRS>
  <OtherCRS>urn:ogc:def:crs:EPSG::3301</OtherCRS>
  <OtherCRS>urn:ogc:def:crs:EPSG::3857</OtherCRS>
  <OtherCRS>urn:ogc:def:crs:EPSG::4326</OtherCRS>
  <ows:WGS84BoundingBox>
    <ows:LowerCorner>-1.0 -1.0</ows:LowerCorner>
    <ows:UpperCorner>0.0 0.0</ows:UpperCorner>
  </ows:WGS84BoundingBox>
</FeatureType>
<FeatureType xmlns:vmk="https://gsavalik.envir.ee/vmk">

```

To get all classifiers, we perform the following request:

https://gsavalik.envir.ee/geoserver/wfs?service=WFS&version=2.0.0&request=GetFeature&type=Name=metsaregister:kl_puuliik&outputFormat=application/json

In the JSON response from the server, we find the code for oak (**tamm**), which is **TA**:

```

{
  "type": "Feature",
  "id": "kl_puuliik.6",
  "geometry": {
    "type": "Polygon",
    "coordinates": [
      []
    ]
  },
  "geometry_name": "shape",
  "properties": {
    "kood": "TA",
    "kirjeldus": "tamm"
  }
},

```

Obtaining the Answer for Task 2.1

We have all the data to answer Task 2.1. We need to create an **AND** filter for the fields **metsaregister:eraldis** with the inventory date and main tree species:

```
<Filter>
  <And>
    <PropertyIsGreaterThanOrEqualTo>
      <PropertyName>invent_kp</PropertyName>
      <Literal>2023-01-01T00:00:00Z</Literal>
    </PropertyIsGreaterThanOrEqualTo>
    <PropertyIsLessThanOrEqualTo>
      <PropertyName>invent_kp</PropertyName>
      <Literal>2023-12-31T23:59:59Z</Literal>
    </PropertyIsLessThanOrEqualTo>
    <PropertyIsEqualTo>
      <PropertyName>peapuuliik_kood</PropertyName>
      <Literal>TA</Literal>
    </PropertyIsEqualTo>
  </And>
</Filter>
```

For simplicity, we assume we are interested in the year 2023 in UTC format and do not adjust for local time.

To get the number of matches from the server, we need to use the argument **resultType=hits**

Combining the layer request, filters, and result type:

https://gsavalik.envir.ee/geoserver/wfs?service=WFS&version=2.0.0&request=GetFeature&typeName=metsaregister:eraldis&outputFormat=application/json&resultType=hits&filter=%3CFilter%3E%3CAnd%3E%3CPropertyIsGreaterThanOrEqualTo%3E%3CPropertyName%3Einvent_kp%3C%2FPropertyName%3E%3CLiteral%3E2023-01-01T00%3A00%3A00Z%3C%2FLiteral%3E%3C%2FPropertyIsGreaterThanOrEqualTo%3E%3CPropertyIsLessThanOrEqualTo%3E%3CPropertyName%3Einvent_kp%3C%2FPropertyName%3E%3CLiteral%3E2023-12-31T23%3A59%3A59Z%3C%2FLiteral%3E%3C%2FPropertyIsLessThanOrEqualTo%3E%3CPropertyIsEqualTo%3E%3CPropertyName%3Epeapuuliik_kood%3C%2FPropertyName%3E%3CLiteral%3ETA%3C%2FLiteral%3E%3C%2FPropertyIsEqualTo%3E%3C%2FAnd%3E%3C%2FFilter%3E

We get the result - **360 plots** as of 16.05.2024 08:26 UTC.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <wfs:FeatureCollection xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:fes="http://www.opengis.net/fes/2.0"
3     xmlns:wfs="http://www.opengis.net/wfs/2.0" xmlns:gml="http://www.opengis.net/gml/3.2"
4     xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:xlink="http://www.w3.org/1999/xlink"
5     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
6     next="https://gsavalik.envir.ee/geoserver/wfs?FILTER=%3CFilter%3EAnd%3E%3CPropertyIsGreaterThanEqualTo
7     %3E%3CPropertyName%3Einvent_kp%3C%2FPropertyName%3E%3CLiteral%3E2023-01-01T00%3A00%3A00Z%3C%2FLiteral%3E%3C%2F
8     PropertyIsGreaterThanEqualTo%3E%3CPropertyIsLessThanEqualTo%3E%3CPropertyName%3Einvent_kp%3C%2F
9     PropertyName%3E%3CLiteral%3E2023-12-31T23%3A59%3A59Z%3C%2FLiteral%3E%3C%2F
10    PropertyIsLessThanEqualTo%3E%3CPropertyIsEqualTo%3E%3CPropertyName%3Eepeapuulik_kood%3C%2F
11    PropertyName%3E%3CLiteral%3ETA%3C%2FLiteral%3E%3C%2FPropertyIsEqualTo%3E%3C%2FAnd%3E%3C%2FFilter%3E&
12    TYPENAME=meisaregister%3Aeraldis&RESULTTYPE=results&REQUEST=GetFeature&OUTPUTFORMAT=application%2Fjson&
13    VERSION=2.0.0&SERVICE=WFS&STARTINDEX=0"
14    numberMatched="360" numberReturned="0" timeStamp="2024-05-16T08:26:11.880Z"
15    xsi:schemaLocation="http://www.opengis.net/wfs/2.0 http://schemas.opengis.net/wfs/2.0/wfs.xsd"/>
```

Task 2.2

For this task, we need the layer **ehak:maakondade_piirid**. First, we will get information about this layer using DescribeFeatureType:

https://gsavalik.envir.ee/geoserver/wfs?service=WFS&version=2.0.0&request=DescribeFeatureType&typeName=ehak:maakondade_piirid

The layer description will be provided in the response:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ehak="kemit.ee/ehak" xmlns:gml="http://www.opengis.net/gml/3.2" xmlns:wfs="http://www.opengis.net/wfs/2.0" schemaLocation="https://gsavalik.envir.ee/geoserver/schemas/gml/3.2.1/gml.xsd"/>
<xsd:complexType name="maakondade_piiridType">
  <xsd:complexContent>
    <xsd:extension base="gml:AbstractFeatureType">
      <xsd:sequence>
        <xsd:element maxOccurs="1" minOccurs="1" name="fid" nillable="false" type="xsd:long"/>
        <xsd:element maxOccurs="1" minOccurs="0" name="shape" nillable="true" type="gml:GeometryPropertyType"/>
        <xsd:element maxOccurs="1" minOccurs="0" name="ehak_kood" nillable="true" type="xsd:string"/>
        <xsd:element maxOccurs="1" minOccurs="0" name="maakond" nillable="true" type="xsd:string"/>
        <xsd:element maxOccurs="1" minOccurs="0" name="tyyp" nillable="true" type="xsd:string"/>
        <xsd:element maxOccurs="1" minOccurs="0" name="vers_algus" nillable="true" type="xsd:dateTime"/>
        <xsd:element maxOccurs="1" minOccurs="0" name="vers_lopp" nillable="true" type="xsd:dateTime"/>
        <xsd:element maxOccurs="1" minOccurs="0" name="seaduslik_alus" nillable="true" type="xsd:string"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="maakondade_piirid" substitutionGroup="gml:AbstractFeature" type="ehak:maakondade_piiridType"/>
</xsd:schema>
```

We need to retrieve all data from this layer except for geographic points. We will use a request that specifies the fields we need:

https://gsavalik.envir.ee/geoserver/wfs?service=WFS&version=2.0.0&request=GetFeature&typeName=ehak:maakondade_piirid&outputFormat=application/json&propertyName=fid,ehak_kood,maakond,tyyp,vers_algus,vers_lopp,seaduslik_alus

This request will return a dataset of counties, from which we need to find Lääne maakond:

```
{
  "type": "Feature",
  "id": "maakondade_piirid.7",
  "geometry": null,
  "properties": {
    "fid": 7,
    "ehak_kood": "0056",
    "maakond": "Lääne maakond",
    "tyyp": "maakond",
    "vers_algus": "2017-11-07T08:40:54Z",
    "vers_lopp": null,
    "seaduslik_alus": "Vormsi vallale uus maakonna kood"
  }
},
```


Python Script

In this task, we cannot perform all operations on the server, so we need to make several requests and process them with code.

1 First Request

Goal: Obtain data on county boundaries.

Action: Send an XML request to the server with the filter **fid=7** (Lääne viru) on the layer **ehak:maakondade_piirid** to get the county boundaries. Save the server response to a file **area_response.xml** for analysis if needed.

```
import requests
from xml.etree import ElementTree as ET
from shapely.geometry import Polygon
import json
import re

def fetch_geo_data():
    # URL and XML for first request (GetFeature)
    url = 'https://gsavalik.envir.ee/geoserver/wfs?service=WFS&version=1.1.0'
    area_query_xml = '''
        <GetFeature service="WFS" version="1.1.0"
            xmlns="http://www.opengis.net/wfs"
            xmlns:gml="http://www.opengis.net/gml"
            xmlns:ogc="http://www.opengis.net/ogc"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://www.opengis.net/wfs
            http://schemas.opengis.net/wfs/1.1.0/wfs.xsd">
            <Query typeName="ehak:maakondade_piirid">
                <ogc:Filter>
                    <ogc:PropertyIsEqualTo>
                        <ogc:PropertyName>fid</ogc:PropertyName>
                        <ogc:Literal>7</ogc:Literal>
                    </ogc:PropertyIsEqualTo>
                </ogc:Filter>
            </Query>
        </GetFeature>
    '''

    headers = {
        'Content-Type': 'text/xml',
        'Accept': 'application/xml'
    }

    # Send first request
    response = requests.post(url, headers=headers, data=area_query_xml)
    if response.status_code != 200:
```

```

        print(f'Error in first request: {response.status_code}')
        return

    # Save response to file
    with open('area_response.xml', 'wb') as file:
        file.write(response.content)

    print("Response for first request saved to area_response.xml")

```

2 Processing the First Response

Goal: Extract the coordinates of all polygons describing the county boundaries.

Action: Extract all **<gml:posList>** elements from the response containing polygon coordinates and collect them into a list.

```

# Parsing coordinates from first response
xml_response = ET.fromstring(response.content)
pos_list_elements =
xml_response.findall(".//{http://www.opengis.net/gml}posList")

if not pos_list_elements:
    print("posList elements not found in the response")
    return

# Making list of polygons
polygons = []
for pos_list_element in pos_list_elements:
    pos_list = pos_list_element.text.strip()
    polygons.append(pos_list)

print(f"Polygons found: {len(polygons)}")

```

3 Second Request:

Goal: Obtain all forest plots intersecting with the county boundaries and matching the given filters.

Action: Form an XML request that includes all county polygons from the first response as intersection conditions (**Intersects**) inside **OR** operator, and send it to the server. Save the formed request in a separate file for possible analysis.

```

# Making conditions with Or operator for all polygons
intersects_conditions = ""
for pos_list in polygons:
    intersects_conditions += f'''
        <ogc:Intersects>
            <ogc:PropertyName>shape</ogc:PropertyName>
            <gml:Polygon>
                <gml:exterior>

```

```

        <gml:LinearRing>
            <gml:posList>{pos_list}</gml:posList>
        </gml:LinearRing>
    </gml:exterior>
</gml:Polygon>
</ogc:Intersects>
'''

# XML for second request (GetFeature with filter)
forest_query_xml = f'''
    <GetFeature service="WFS" version="1.1.0"
        xmlns="http://www.opengis.net/wfs"
        xmlns:gml="http://www.opengis.net/gml"
        xmlns:ogc="http://www.opengis.net/ogc"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.opengis.net/wfs
        http://schemas.opengis.net/wfs/1.1.0/wfs.xsd">
        <Query typeName="metsaregister:eraldis">
            <ogc:Filter>
                <ogc:And>
                    <ogc:PropertyIsGreaterThanOrEqualTo>
                        <ogc:PropertyName>invent_kp</ogc:PropertyName>
                        <ogc:Literal>2023-01-01T00:00:00Z</ogc:Literal>
                    </ogc:PropertyIsGreaterThanOrEqualTo>
                    <ogc:PropertyIsLessThanOrEqualTo>
                        <ogc:PropertyName>invent_kp</ogc:PropertyName>
                        <ogc:Literal>2023-12-31T23:59:59Z</ogc:Literal>
                    </ogc:PropertyIsLessThanOrEqualTo>
                    <ogc:PropertyIsEqualTo>
                        <ogc:PropertyName>peapuuliik_kood</ogc:PropertyName>
                        <ogc:Literal>TA</ogc:Literal>
                    </ogc:PropertyIsEqualTo>
                <ogc:Or>
                    {intersects_conditions}
                </ogc:Or>
            </ogc:And>
        </ogc:Filter>
    </Query>
</GetFeature>
'''

# Saving second request XML to file
with open('forest_query.xml', 'w') as file:
    file.write(forest_query_xml)

print("Second request saved to forest_query.xml")

```

4 Processing the Second Response

Goal: Extract the coordinates of forest plot polygons and calculate their centroids.

Action: Send the request to the server and save the response to a file, extract all **<gml:posList>** elements, split them into coordinates, create polygons, and calculate their centroids. We use the Python library shapely to find centroids of the polygons. Save the centroids to a JSON file for possible future analysis.

```
# Sending second request for getting all forest areas
response = requests.post(url, headers=headers, data=forest_query_xml)
if response.status_code != 200:
    print(f'Error in second request: {response.status_code}')
    return

# Saving response to file
with open('forest_response.xml', 'wb') as file:
    file.write(response.content)

print("Second response saved to forest_response.xml")

# Processing response for getting forest areas and their centroids
xml_response = response.content.decode('utf-8')
eraldis_elements = re.findall(r'(<metsaregister:eraldis
.*?</metsaregister:eraldis>)', xml_response, re.DOTALL)

# Checking XML structure and printing amount of records
feature_count = len(eraldis_elements)
print('Feature members:', feature_count)

# Centroids coordinates list
centroids = []

for eraldis_element in eraldis_elements:
    pos_list_elements = re.findall(r'(<gml:posList>(.*?)</gml:posList>)',
    eraldis_element, re.DOTALL)
    for pos_list_element in pos_list_elements:
        pos_list = pos_list_element.strip().split()
        coordinates = [(float(pos_list[i]), float(pos_list[i + 1])) for i in
        range(0, len(pos_list), 2)]
        if len(coordinates) > 2: # We should make sure that there is at least 3
        points to make polygon
            polygon = Polygon(coordinates)
            centroid = polygon.centroid
            centroids.append(centroid)
        else:
            print("Not enough coordinates to form a polygon:", coordinates)
```

```

print('Centroids amount:', len(centroids))

# Saving centroids to file for further analysis
with open('centroids.json', 'w') as f:
    centroids_data = [{"x": centroid.x, "y": centroid.y} for centroid in
centroids]
    json.dump(centroids_data, f, indent=4)

print("Centroids saved to centroids.json")

```

5 Third Request

Goal: Determine which county each forest plot centroid falls into and display the data in the console.

Action: For each centroid, form an XML request with the Intersects filter and send it to the server. Extract the county name from the response, compare it with the required one, and display the result in the console. We use the Python library shapely to find centroids of the polygons.

```

# Making request to check centroids intersects with maakond areas
intersect_count = 0

for centroid in centroids:
    intersection_query_xml = f'''
        <GetFeature service="WFS" version="1.1.0"
            xmlns="http://www.opengis.net/wfs"
            xmlns:gml="http://www.opengis.net/gml"
            xmlns:ogc="http://www.opengis.net/ogc"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://www.opengis.net/wfs
http://schemas.opengis.net/wfs/1.1.0/wfs.xsd">
            <Query typeName="ehak:maakondade_piirid">
                <PropertyName>fid</PropertyName>
                <PropertyName>maakond</PropertyName>
                <ogc:Filter>
                    <ogc:Intersects>
                        <ogc:PropertyName>shape</ogc:PropertyName>
                        <gml:Point srsName="urn:ogc:def:crs:EPSG::3301">
                            <gml:pos>{centroid.x} {centroid.y}</gml:pos>
                        </gml:Point>
                    </ogc:Intersects>
                </ogc:Filter>
            </Query>
        </GetFeature>
    '''

    response = requests.post(url, headers=headers, data=intersection_query_xml)
    if response.status_code == 200:
        xml_response = response.content.decode('utf-8')

```

```

        matches = re.findall(r'<ehak:maakond>(.*?)</ehak:maakond>',
xml_response)
        if matches and 'Lääne maakond':
            intersect_count += 1
            for match in matches:
                print(f'Centroid {centroid.x}, {centroid.y} located in maakond:
{match}')

print(f'Amount of centroids that located in Lääne maakond: {intersect_count}')

```

Answer to Task 2.2

After running the script, we see the progress and result of the task in the console:

Polygons found: 308

Second request saved to forest_query.xml

Second response saved to forest_response.xml

Feature members: 20

Centroids amount: 20

Centroids saved to centroids.json

Centroid 6520162.769933613, 501292.4974884256 located in maakond: Lääne maakond

Centroid 6530966.26742345, 470565.1546613222 located in maakond: Lääne maakond

Centroid 6543873.754939496, 486756.6057401718 located in maakond: Lääne maakond

Centroid 6522375.728491703, 490741.8366681041 located in maakond: Lääne maakond

Centroid 6535268.946059849, 491955.9673894772 located in maakond: Lääne maakond

Centroid 6535275.543743414, 494244.8019264315 located in maakond: Lääne maakond

Centroid 6534311.157929512, 478586.7653442363 located in maakond: Lääne maakond

Centroid 6523240.413769688, 471516.9218962486 located in maakond: Lääne maakond

Centroid 6523197.898790737, 471520.50048805843 located in maakond: Lääne maakond

Centroid 6531360.086362745, 477175.40014732117 located in maakond: Lääne maakond

Centroid 6529283.646480764, 498760.7762148661 located in maakond: Lääne maakond

Centroid 6543729.599085709, 486650.25060556666 located in maakond: Lääne maakond

Centroid 6539170.916664042, 494516.3686427767 located in maakond: Lääne maakond

Centroid 6545394.650315111, 486883.7528621549 located in maakond: Lääne maakond

Centroid 6517831.167998004, 473951.34258544765 located in maakond: Lääne maakond

Centroid 6530940.102837833, 476129.76093893807 located in maakond: Lääne maakond

Centroid 6531114.058815941, 475947.9544449159 located in maakond: Lääne maakond

Centroid 6531205.589462678, 495886.7358337739 located in maakond: Lääne maakond

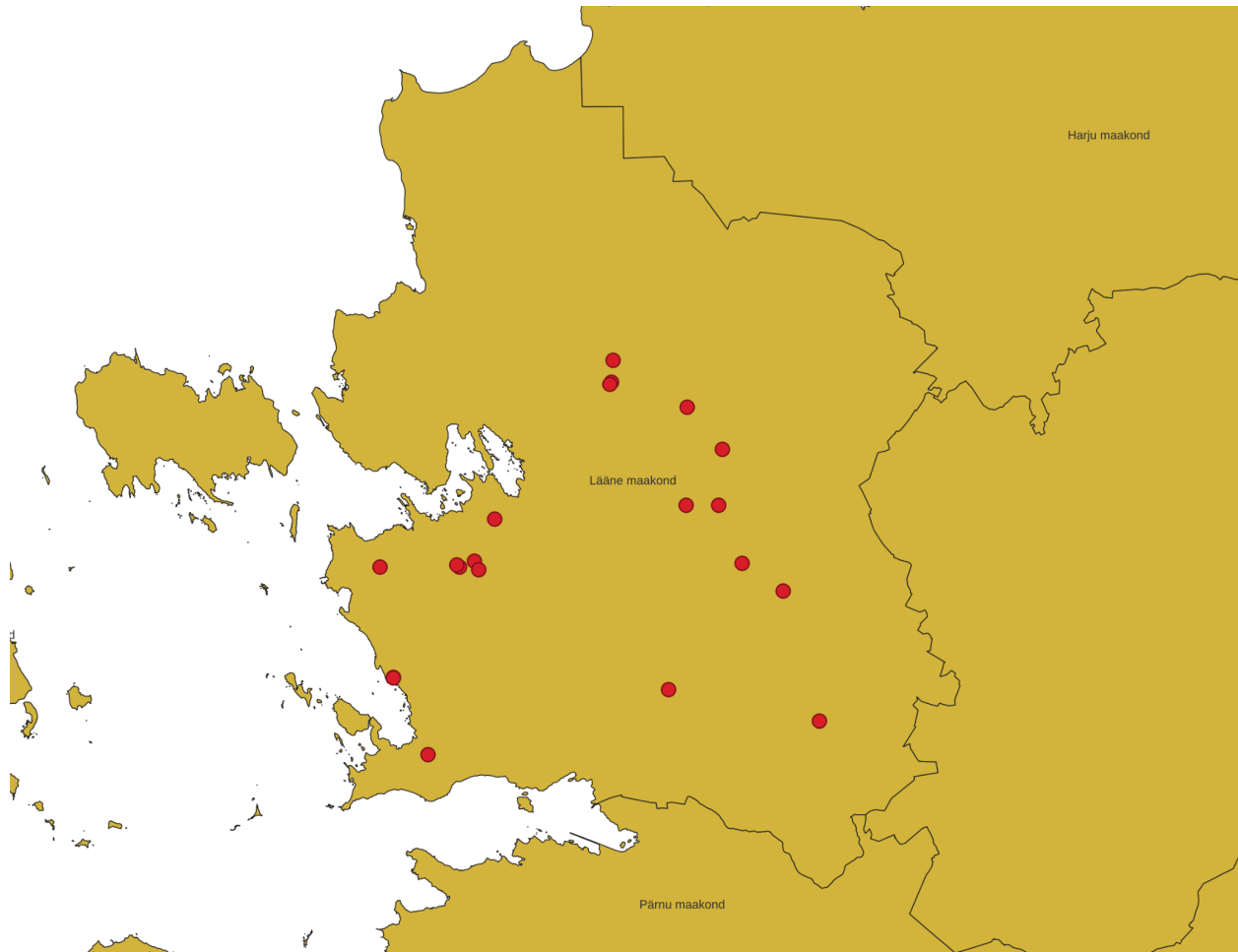
Centroid 6542132.582232924, 492058.4228261494 located in maakond: Lääne maakond

Centroid 6530768.250635989, 477494.38518613 located in maakond: Lääne maakond

Amount of centroids that located in Lääne maakond: 20

Result: 20 plots inventoried in 2023, with oak as the main tree species, have their central points in the Lääne-Viru region. The request time is 2024-05-17T13:12:30.086Z.

Visualization on Map:

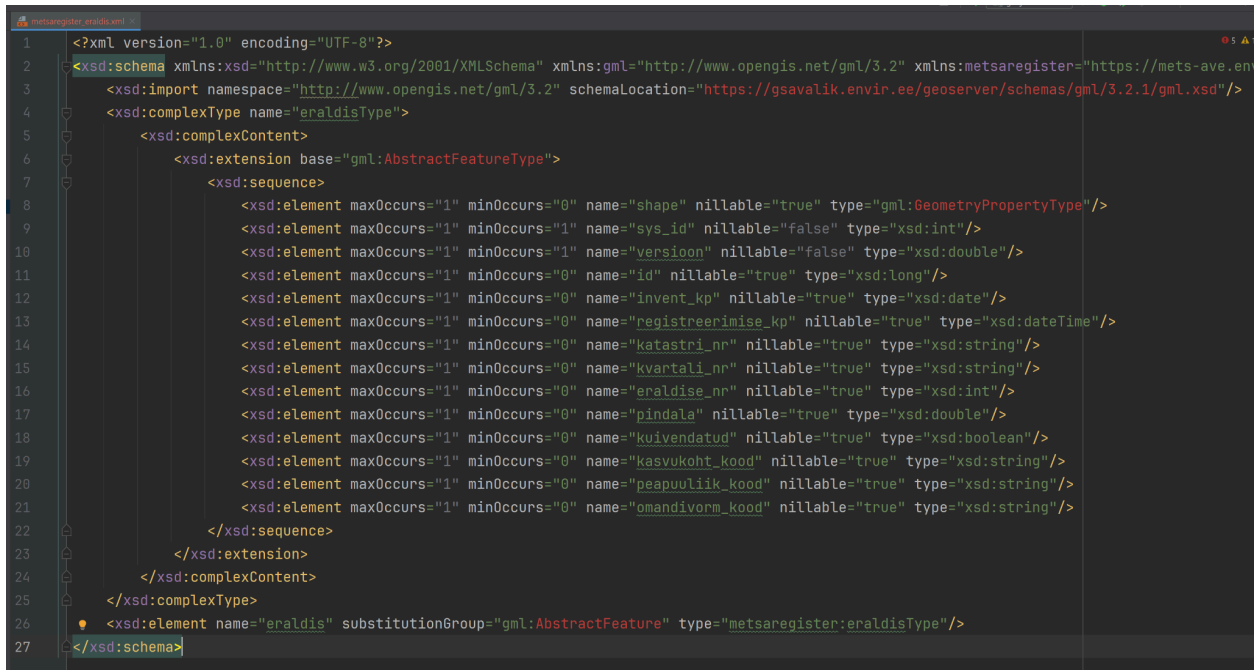


You can download and run the code from the repository:

<https://github.com/DannPeterson/geoserver-assignment/blob/main/task2.2.py>

Task 2.3

Let's look again at the description of the layer **metsaregister:eraldis**



We are interested in **pindala**.

Since the envir GeoServer does not support summing functions and does not have an OWS service for these operations, we need to retrieve all pindala data from Task 2.1 and sum them using a script.

Python Script

1 Data Request

Goal: Obtain area data for specific forest plots.

Action: Formulate a request to the server similar to Task 2.1, but limit the selection by the **pindala** field. Send the request and save the response to a file for possible further analysis.

```
url = 'https://gsavalik.envir.ee/geoserver/wfs?service=WFS&version=1.1.0'
area_query_xml = '''
  <GetFeature service="WFS" version="1.1.0"
    xmlns="http://www.opengis.net/wfs"
    xmlns:gml="http://www.opengis.net/gml"
    xmlns:ogc="http://www.opengis.net/ogc"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:wfs="http://www.opengis.net/wfs"
    xsi:schemaLocation="http://www.opengis.net/wfs
      http://schemas.opengis.net/wfs/1.1.0/wfs.xsd">
    <wfs:Query typeName="metsaregister:eraldis" srsName="EPSG:4326">
```



```

        <wfs:PropertyName>metsaregister:pindala</wfs:PropertyName>
        <ogc:Filter>
            <ogc:And>
                <ogc:PropertyIsGreaterThanOrEqualTo>
                    <ogc:PropertyName>invent_kp</ogc:PropertyName>
                    <ogc:Literal>2023-01-01T00:00:00Z</ogc:Literal>
                </ogc:PropertyIsGreaterThanOrEqualTo>
                <ogc:PropertyIsLessThanOrEqualTo>
                    <ogc:PropertyName>invent_kp</ogc:PropertyName>
                    <ogc:Literal>2023-12-31T23:59:59Z</ogc:Literal>
                </ogc:PropertyIsLessThanOrEqualTo>
                <ogc:PropertyIsEqualTo>
                    <ogc:PropertyName>peapuuliik_kood</ogc:PropertyName>
                    <ogc:Literal>TA</ogc:Literal>
                </ogc:PropertyIsEqualTo>
            </ogc:And>
        </ogc:Filter>
    </wfs:Query>
</GetFeature>
'''

headers = {
    'Content-Type': 'text/xml',
    'Accept': 'application/xml'
}

# Send the request
response = requests.post(url, headers=headers, data=area_query_xml)
if response.status_code != 200:
    print(f'Error in request: {response.status_code}')
    return

# Save the response to a file
with open('sum_response.xml', 'wb') as file:
    file.write(response.content)

```

2 Data Analysis

Goal: Process the server response and obtain area data.

Action: Split the server response into individual plots, iterate through the data array, and sum all the areas.

```

# Parse the XML response to sum up 'pindala' values
tree = ET.fromstring(response.content)
namespaces = {
    'gml': "http://www.opengis.net/gml",
    'metsaregister': "https://mets-ave.envir.ee"
}

```

```
# Find all 'pindala' elements
pindala_elements = tree.findall('./metsaregister:pindala', namespaces)
print(f'Total elements: {len(pindala_elements)}')
pindala_values = [float(pindala.text) for pindala in pindala_elements]

# Calculate total and average 'pindala'
total_pindala = sum(pindala_values)

print(f'Total area: {total_pindala}')
```

Answer to Task 2.3

After running the script, we see the progress and result of the task in the console:

```
Total elements: 360
```

```
Total area: 472.13
```

Result: The total area of the plots from Task 2.1 is 472.13 hectares. The request time is 2024-05-20T05:46:28.890Z.

You can download and run the code from the repository:

<https://github.com/DannPeterson/geoserver-assignment/blob/main/task2.3.py>

Task 2.4

For this task, we need the layer **met saregister:teatis** with active notifications. Let's look at the layer description:

```
teatis1.xml x
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:gml="http://www.opengis.net/gml/3.2" xmlns:met saregister="https://
3  <xsd:import namespace="http://www.opengis.net/gml/3.2" schemaLocation="https://gsavalik.envir.ee/geoserver/schemas/gml/3.2.1/
4  <xsd:complexType name="teatisType">
5  <xsd:complexContent>
6  <xsd:extension base="gml:AbstractFeatureType">
7  <xsd:sequence>
8  <xsd:element maxOccurs="1" minOccurs="0" name="shape" nillable="true" type="gml:GeometryPropertyType"/>
9  <xsd:element maxOccurs="1" minOccurs="1" name="sys_id" nillable="false" type="xsd:int"/>
10 <xsd:element maxOccurs="1" minOccurs="0" name="teatise_nr" nillable="true" type="xsd:string"/>
11 <xsd:element maxOccurs="1" minOccurs="0" name="kinnistu_nimetuse" nillable="true" type="xsd:string"/>
12 <xsd:element maxOccurs="1" minOccurs="0" name="kinnistu_nr" nillable="true" type="xsd:long"/>
13 <xsd:element maxOccurs="1" minOccurs="0" name="met skond" nillable="true" type="xsd:string"/>
14 <xsd:element maxOccurs="1" minOccurs="0" name="katastri_nr" nillable="true" type="xsd:string"/>
15 <xsd:element maxOccurs="1" minOccurs="0" name="kvartali_nr" nillable="true" type="xsd:string"/>
16 <xsd:element maxOccurs="1" minOccurs="0" name="eraldis_nr" nillable="true" type="xsd:int"/>
17 <xsd:element maxOccurs="1" minOccurs="0" name="pindala" nillable="true" type="xsd:double"/>
18 <xsd:element maxOccurs="1" minOccurs="0" name="too_kood" nillable="true" type="xsd:string"/>
19 <xsd:element maxOccurs="1" minOccurs="0" name="raiutav_maht" nillable="true" type="xsd:int"/>
20 <xsd:element maxOccurs="1" minOccurs="0" name="otsus" nillable="true" type="xsd:string"/>
21 <xsd:element maxOccurs="1" minOccurs="0" name="otsuse_pohjendus" nillable="true" type="xsd:string"/>
22 <xsd:element maxOccurs="1" minOccurs="0" name="otsus_kinnitatus_kp" nillable="true" type="xsd:dateTime"/>
23 <xsd:element maxOccurs="1" minOccurs="0" name="kehtiv_kuni" nillable="true" type="xsd:date"/>
24 </xsd:sequence>
25 </xsd:extension>
26 </xsd:complexContent>
27 </xsd:complexType>
28 <xsd:element name="teatis" substitutionGroup="gml:AbstractFeature" type="met saregister:teatisType"/>
29 </xsd:schema>
```

We need to collect the geometry of all forest plots from Task 2.1 and check which notification plots they intersect with. For this, we will use a Python script.

Python Script

1 Request for Forest Plots from Task 2.1

Goal: Obtain data on the areas of specific forest plots.

Action: Formulate a request to the server similar to Task 2.1 and save the server response to a file.

```
def fetch_geo_data():
    # URL and XML for first request (GetFeature)
    url = 'https://gsavalik.envir.ee/geoserver/wfs?service=WFS&version=1.1.0'
    area_query_xml = ''
    <GetFeature service="WFS" version="1.1.0"
        xmlns="http://www.opengis.net/wfs"
        xmlns:gml="http://www.opengis.net/gml"
        xmlns:ogc="http://www.opengis.net/ogc"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```

xsi:schemaLocation="http://www.opengis.net/wfs
http://schemas.opengis.net/wfs/1.1.0/wfs.xsd">
<Query typeName="metsaregister:eraldis">
  <ogc:Filter>
    <ogc:And>
      <ogc:PropertyIsGreaterThanOrEqualTo>
        <ogc:PropertyName>invent_kp</ogc:PropertyName>
        <ogc:Literal>2023-01-01T00:00:00Z</ogc:Literal>
      </ogc:PropertyIsGreaterThanOrEqualTo>
      <ogc:PropertyIsLessThanOrEqualTo>
        <ogc:PropertyName>invent_kp</ogc:PropertyName>
        <ogc:Literal>2023-12-31T23:59:59Z</ogc:Literal>
      </ogc:PropertyIsLessThanOrEqualTo>
      <ogc:PropertyIsEqualTo>
        <ogc:PropertyName>peapuuliik_kood</ogc:PropertyName>
        <ogc:Literal>TA</ogc:Literal>
      </ogc:PropertyIsEqualTo>
    </ogc:And>
  </ogc:Filter>
</Query>
</GetFeature>
'''

headers = {
    'Content-Type': 'text/xml',
    'Accept': 'application/xml'
}

# Send first request
response = requests.post(url, headers=headers, data=area_query_xml)
if response.status_code != 200:
    print(f'Error in first request: {response.status_code}')
    return

# Save response to file
with open('forest_areas_response.xml', 'wb') as file:
    file.write(response.content)

print("Response for first request saved to forest_areas_response.xml")

```

2 Processing Forest Plot Polygons

Goal: Extract coordinates of forest plot polygons and add a buffer.

Action: Extract all forest plot elements from **<gml:posList>**, split them into coordinates, and create polygons.

Polygons of notifications can intersect with forest plot polygons by very small amounts, from thousandths of a millimeter to a few centimeters. We will consider this an error and will not count it as an active notification on the plot. For this, we use the Shapely library to add a

20-centimeter buffer to the forest plot polygons, reducing their size and eliminating unnecessary intersections. Collect all polygons into a list.

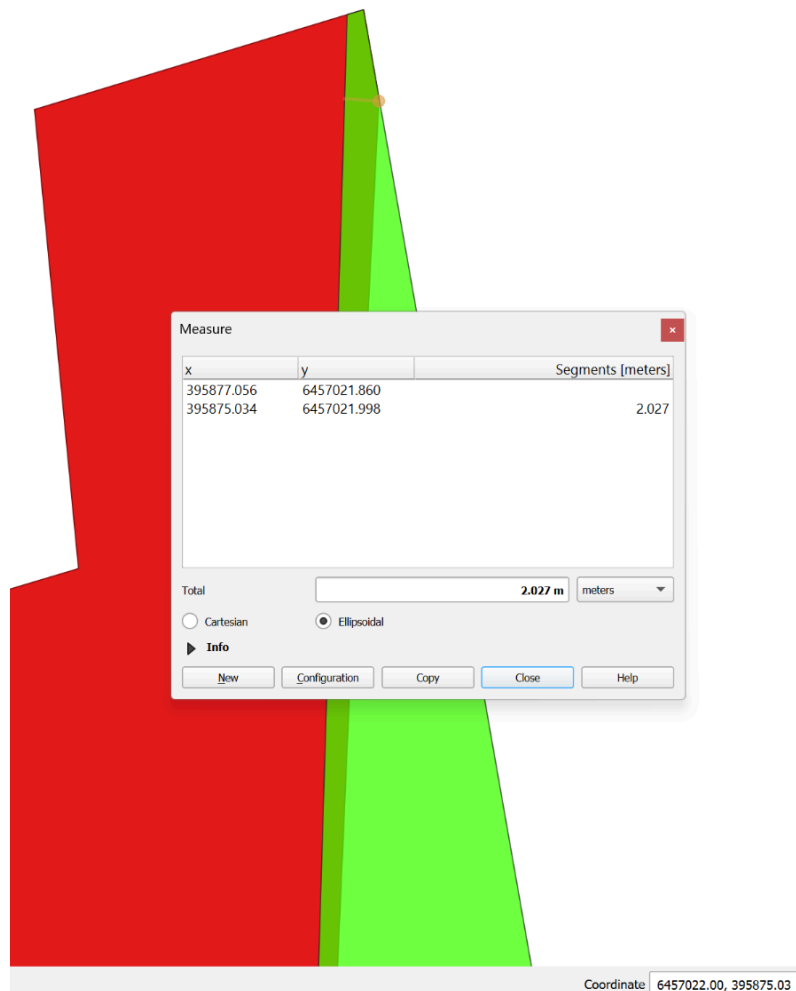
```
# Parsing coordinates from first response
xml_response = ET.fromstring(response.content)
pos_list_elements =
xml_response.findall(".//{http://www.opengis.net/gml}posList")

if not pos_list_elements:
    print("posList elements not found in the response")
    return

# Making list of polygons
buffer_distance = -0.2 # 20cm buffer
polygons = []
for pos_list_element in pos_list_elements:
    pos_list = pos_list_element.text.strip().split()
    coordinates = [(float(pos_list[i]), float(pos_list[i + 1])) for i in
range(0, len(pos_list), 2)]
    if len(coordinates) > 2:
        polygon = Polygon(coordinates).buffer(buffer_distance)
        polygons.append(polygon)

print(f"Polygons found: {len(polygons)}")
```

Some notifications and forest plots overlap by several tens of centimeters to two meters - we will count such overlaps as active notifications on the plot:



3 Creating a Request for the **metasregister:teatis** Layer with Forest Plot Geodata:

Goal: Formulate a request for the notification layer.

Action: Iterate through the list of collected and processed forest plot coordinates and create filters from them.

Overlaps: When a notification plot intersects with a forest plot.

Within: When a forest plot is inside a notification plot.

Contains: When a forest plot contains a notification plot.

To determine if there is an active notification on the forest plot, any of these matches will suffice, so we use an OR filter.

Formulate the request in a file for possible analysis and send the request to the server.

```
# Making conditions with Or operator for all polygons
intersects_conditions = ""
for polygon in polygons:
    pos_list = " ".join([f"{coord[0]} {coord[1]}" for coord in
        polygon.exterior.coords])
```

```

intersects_conditions += f'''
    <ogc:Overlaps>
        <ogc:PropertyName>shape</ogc:PropertyName>
        <gml:Polygon>
            <gml:exterior>
                <gml:LinearRing>
                    <gml:posList>{pos_list}</gml:posList>
                </gml:LinearRing>
            </gml:exterior>
        </gml:Polygon>
    </ogc:Overlaps>
    <ogc:Within>
        <ogc:PropertyName>shape</ogc:PropertyName>
        <gml:Polygon>
            <gml:exterior>
                <gml:LinearRing>
                    <gml:posList>{pos_list}</gml:posList>
                </gml:LinearRing>
            </gml:exterior>
        </gml:Polygon>
    </ogc:Within>
    <ogc:Contains>
        <ogc:PropertyName>shape</ogc:PropertyName>
        <gml:Polygon>
            <gml:exterior>
                <gml:LinearRing>
                    <gml:posList>{pos_list}</gml:posList>
                </gml:LinearRing>
            </gml:exterior>
        </gml:Polygon>
    </ogc:Contains>
'''

# XML for second request (GetFeature with filter)
forest_query_xml = f'''
    <GetFeature service="WFS" version="1.1.0"
        xmlns="http://www.opengis.net/wfs"
        xmlns:gml="http://www.opengis.net/gml"
        xmlns:ogc="http://www.opengis.net/ogc"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.opengis.net/wfs
        http://schemas.opengis.net/wfs/1.1.0/wfs.xsd">
        <Query typeName="metsaregister:teatis">
            <ogc:Filter>
                <ogc:Or>
                    {intersects_conditions}
                </ogc:Or>
            </ogc:Filter>
        </Query>
'''

```

```

    </GetFeature>
'''

# Saving second request XML to file
with open('teatis_query.xml', 'w') as file:
    file.write(forest_query_xml)

print("Second request saved to teatis_query.xml")

# Sending second request for getting all relevant areas
response = requests.post(url, headers=headers, data=forest_query_xml)
if response.status_code != 200:
    print(f'Error in second request: {response.status_code}')
    return

# Saving response to file
with open('overlaps_within_contains_buffer_20cm.xml', 'wb') as file:
    file.write(response.content)

print("Second response saved to overlaps_within_contains_buffer_20cm.xml")

```

4 Analyzing the Second Request Response and Counting Notifications:

Goal: Obtain the number of notifications on forest plots.

Action: Collect the number of `<metsaregister:otsus_kinnitatus_kp>` tags, equate them to the number of active notifications, and display the result in the console.

```

# Processing response for getting relevant areas
xml_response = response.content.decode('utf-8')
teatis_elements = re.findall(r'(<metsaregister:teatis
.*?</metsaregister:teatis>)', xml_response, re.DOTALL)

# Extract and count valid notifications
valid_notifications_count = 0
for teatis in teatis_elements:
    otsus_kinnitatus_kp =
re.search(r'<metsaregister:otsus_kinnitatus_kp>(.*?)</metsaregister:otsus_kinni
tatus_kp>', teatis)
    if otsus_kinnitatus_kp:
        valid_notifications_count += 1

# Print the result
print(f'Valid notifications count: {valid_notifications_count}')

```

Answer to Task 2.4

After running the script, we see the progress and result of the task in the console:

Response for first request saved to forest_areas_response.xml

Polygons found: 372

Second request saved to teatis_query.xml

Second response saved to overlaps_within_contains_buffer_20cm.xml

Valid notifications count: 36

Result: The number of active notifications on plots from Task 2.1 is 36. The request time is 2024-05-21T06:34:26.583Z.

You can download and run the code from the repository.

<https://github.com/DannPeterson/geoserver-assignment/blob/main/task2.4.py>