Objetivos:

- Carregar e visualizar arquivos raster no QGIS utilizando PyQGIS;
- Explorar propriedades e extrair estatísticas de bandas raster;
- Aplicar técnicas de recorte de imagens raster por extensão e por máscara;
- Manipular valores de pixels em imagens raster com PyQGIS;
- Criar e exportar novas imagens raster utilizando PyQGIS e GDAL.

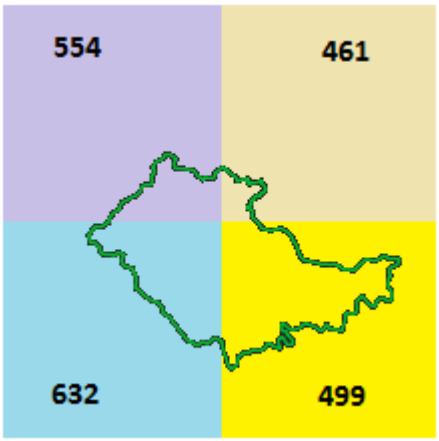
I. Descrição dos dados

Quando trabalhamos com dados raster no QGIS, frequentemente precisamos compreender a estrutura e o conteúdo das imagens utilizadas, especialmente ao manipular subconjuntos espaciais, analisar valores de atributos ou realizar operações de processamento.

Os arquivos utilizados nesta aula são imagens raster no formato GeoTIFF (*Tagged Image File Format*). Nos exemplos serão utilizadas as imagens armazenadas na pasta "trmm". Essa pasta contém 13 imagens com dados de precipitação acumulada mensal provenientes da missão TRMM – Tropical Rainfall Measuring Mission, cobrindo o período de setembro de 2013 a setembro de 2014. Essas imagens foram gentilmente cedidas pelo Pesquisador Dr. Egidio Arai (link).

Sobre as imagens:

- Cada imagem contém valores de precipitação acumulada ao longo de um mês. Por exemplo, o arquivo "2013_09.tif" corresponde à precipitação acumulada no mês de setembro de 2013;
- Cada pixel dessas imagens representa uma área de 0,25° x 0,25°, o que equivale, aproximadamente, a 27 km x 27 km na superfície terrestre;
- No exemplo a seguir, é apresentado o polígono do município de Jacareí, sobreposto aos quatro pixels da imagem "2013_09.tif" que cobrem a área deste município;
- Os valores apresentados neste exemplo correspondem à precipitação acumulada em cada pixel referente ao mês de setembro de 2013.



II. Carregar um arquivo TIF com PyQGIS

Nesta seção, será apresentado como carregar imagens raster no formato GeoTIFF utilizando o PyQGIS, que é a API de automação e extensão do QGIS baseada na linguagem Python.

Esse procedimento é fundamental para automatizar tarefas repetitivas, integrar fluxos de processamento ou desenvolver plugins personalizados no QGIS.

Como exemplo, será carregado um arquivo TIF que contém dados de precipitação acumulada para o mês de setembro de 2013. O objetivo é demonstrar os comandos necessários para importar o arquivo como uma camada raster no ambiente do QGIS, de modo que ela possa ser visualizada e analisada juntamente com outras informações geoespaciais.

Exemplo 1: Carregar um arquivo TIF como camada do QGIS usando PyQGIS.

```
# Caminho do arquivo TIF
caminho = 'D:/pasta/trmm/2013_09.tif'

# Carregar a camada raster
camada = QgsRasterLayer(caminho, 'imagem')

# Adiciona a camada ao projeto QGIS
QgsProject.instance().addMapLayer(camada)
```

III. Explorar propriedades da camada raster

Após carregar a camada raster, é fundamental explorar e compreender algumas de suas principais propriedades.

Entre as propriedades mais relevantes, destacam-se:

- Número de bandas: um raster pode conter uma ou múltiplas bandas. Cada banda corresponde a um conjunto de valores associados a cada pixel. No caso das imagens de precipitação da missão TRMM, cada arquivo possui apenas uma banda, contendo os valores acumulados mensais de precipitação;
- Extensão espacial: define os limites geográficos da camada raster, informando as coordenadas mínimas e máximas nos eixos X e Y. Essa informação é essencial para compreender a cobertura espacial da imagem e para realizar operações de recorte, sobreposição ou integração com outras camadas;
- Sistema de Referência Espacial (CRS): especifica o sistema de coordenadas utilizado para georreferenciar os dados raster. O conhecimento do CRS é fundamental para garantir que todas as camadas do projeto estejam alinhadas espacialmente e para a correta interpretação dos dados.

O PyQGIS permite acessar essas informações utilizando os seguintes comandos:

```
# Número de bandas
print("Número de bandas:", camada.bandCount())

# Resolução espacial
print("Resolução:",camada.rasterUnitsPerPixelX())
```

```
# Extensão espacial
print("Extensão:", camada.extent())

# Sistema de referência espacial (CRS)
print("CRS:", camada.crs().authid())
```

Resultado do processamento:

```
Número de bandas: 1
Resolução: 0.25
Extensão: <QgsRectangle: -80 -50, -30 10>
CRS: EPSG:4326
```

IV. Extrair estatísticas de uma banda

A extração de estatísticas descritivas de uma banda raster é uma etapa fundamental no processo de análise espacial. Por meio desse procedimento, é possível obter um panorama geral sobre a distribuição dos valores presentes na imagem, facilitando a identificação de padrões, tendências ou anomalias.

No contexto das imagens de precipitação da missão TRMM, as estatísticas da banda incluem informações como:

- Valor mínimo: menor valor registrado na banda, indicando a menor quantidade acumulada de precipitação no período;
- Valor máximo: maior valor registrado, correspondendo à maior quantidade de precipitação acumulada;
- Média: valor médio da precipitação acumulada, útil para análises comparativas ou para a caracterização geral da área de estudo;
- Desvio padrão: medida de dispersão que indica o grau de variação dos valores de precipitação em relação à média.

A obtenção dessas estatísticas pode ser realizada de forma automatizada com o PyQGIS, por meio do provedor de dados associado à camada raster:

```
# Obtém provedor de dados
provedor = camada.dataProvider()

# Estatísticas da banda 1
estatisticas = provedor.bandStatistics(1)

print("Mínimo:", estatisticas.minimumValue)
print("Máximo:", estatisticas.maximumValue)
print("Média:", estatisticas.mean)
print("Desvio padrão:", estatisticas.stdDev)
```

Resultado do processamento:

```
Mínimo: 0.0
Máximo: 10595.0
Média: 849.9015
Desvio padrão: 954.8560715321787
```

V. Recortar (clip) o raster por extensão

O recorte (ou clip) de dados raster é uma operação essencial quando se deseja limitar a análise ou o processamento a uma determinada área de interesse, reduzindo o volume de dados e focando em uma região específica.

Neste exemplo, será realizada a operação de recorte sobre a imagem de precipitação acumulada referente ao mês de setembro de 2013, delimitando-a pelos seguintes limites geográficos:

- Extremo oeste (esquerdo): -52
- Extremo leste (direito): -39
- Extremo sul (inferior): -25
- Extremo norte (superior): -18

Esses limites definem a extensão espacial do recorte, permitindo gerar um novo arquivo raster correspondente apenas à área especificada.

O PyQGIS, por meio da interface com os algoritmos do GDAL, permite realizar essa operação utilizando o comando `gdal:cliprasterbyextent`. É importante destacar que o parâmetro `PROJWIN` deve ser fornecido como uma string contendo quatro valores numéricos separados por vírgulas, representando os limites da extensão no seguinte formato: xmin, xmax, ymin, ymax.

O comando para realizar o recorte é apresentado a seguir:

```
# Definir a extensão de recorte:
# xmin, xmax, ymin, ymax
extensao = '-52, -39, -25, -18'

saida = 'D:/pasta/trmm/recorte.tif'

processing.run("gdal:cliprasterbyextent", {
    'INPUT': camada,
    'PROJWIN': extensao,
    'OUTPUT': saida
})
```

O resultado será um novo arquivo no formato GeoTIFF, contendo exclusivamente a área delimitada.

VI. Recortar usando máscara

Esse tipo de recorte, conhecido como "clip by mask layer", é útil quando se deseja limitar a área de um raster conforme os limites precisos de uma feição vetorial, em vez de recortar com base em uma extensão retangular.

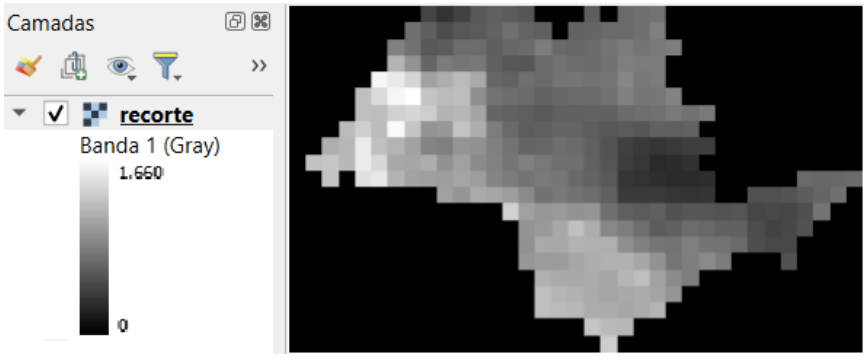
No QGIS, esse procedimento pode ser automatizado utilizando o PyQGIS em conjunto com o algoritmo `gdal:cliprasterbymasklayer`, que permite realizar o recorte de um raster com base em uma camada vetorial, geralmente no formato Shapefile (.shp), GeoPackage (.gpkg), ou outros formatos compatíveis.

Exemplo de código em PyQGIS:

```
caminho_vetor = 'D:/pasta/uf.gpkg'
```

```
caminho_raster = 'D:/pasta/trmm/2013_09.tif'
caminho_saida = 'D:/pasta/trmm/recorte.tif'

# Carrega a camada vetorial
camada_uf = QgsVectorLayer(caminho_vetor, 'uf', 'ogr')

# Cria um QgsFeatureRequest para filtrar apenas São Paulo
request = QgsFeatureRequest().setFilterExpression("uf = 'SP'")

# Obtém a feição de SP
feicao_sp = next(camada_uf.getFeatures(request))

# Cria uma camada temporária em memória com apenas SP
camada_sp = QgsVectorLayer("Polygon?crs=EPSG:4326", "sp", "memory")
# Adiciona a feição na camada temporária
camada_sp.dataProvider().addFeature(feicao_sp)

# Carrega a camada raster
camada_raster = QgsRasterLayer(caminho_raster, 'imagem')

# Executa o recorte utilizando a máscara vetorial
processing.run("gdal:cliprasterbymasklayer", {
    'INPUT': camada_raster,
    'MASK': camada_sp,
    'OUTPUT': caminho_saida,
    'CROP_TO_CUTLINE': True  # Recorta na extensão exata da máscara
})
```

Resultado do processamento:



VII. Percorrer os pixels de um raster com PyQGIS

O processamento e a análise de dados raster muitas vezes requerem o acesso individual aos valores de cada pixel. No PyQGIS, esse procedimento pode ser realizado utilizando o provedor de dados (`dataProvider`) da camada raster, juntamente com a classe `QgsRasterBlock`.

Essa abordagem permite, por exemplo, aplicar filtros, realizar cálculos estatísticos personalizados ou transformar valores conforme regras específicas.

Exemplo 2: Percorrer os pixels da imagem de setembro de 2013 e imprimir no console os pixels com precipitação acumulada no intervalo [5000, 5050] mm.

```
# Caminho do arquivo raster
caminho_raster = 'D:/pasta/trmm/2013_09.tif'

# Carrega a camada raster
camada = QgsRasterLayer(caminho_raster, 'imagem')

# Obtém o provedor de dados
provedor = camada.dataProvider()

# Define a banda a ser acessada
banda = 1
# Obtém a extensão da camada
extensao = camada.extent()
# Obtém o tamanho dos pixels (resolução)
resolucao_x = camada.rasterUnitsPerPixelX()
resolucao_y = camada.rasterUnitsPerPixelY()

# Calcula o número de colunas e linhas
colunas = int(extensao.width() / resolucao_x)
linhas = int(extensao.height() / resolucao_y)

print(f"Raster com {linhas} linhas e {colunas} colunas.")

# Obtém o bloco de dados raster para a banda e extensão
bloco = provedor.block(banda, extensao, colunas, linhas)

# Percorre as linhas
for linha in range(linhas):
    # Percorre os pixels na linha
    for coluna in range(colunas):
        # Obtém o valor do pixel
        valor = bloco.value(coluna, linha)
        if valor >= 5000 and valor <= 5050:
            print(f"Pixel ({linha}, {coluna}): {valor}")
```

Resultado do processamento:

```
Raster com 240 linhas e 200 colunas.
Pixel (11, 23): 5019.0
Pixel (25, 38): 5003.0
Pixel (40, 30): 5050.0
Pixel (41, 30): 5005.0
Pixel (59, 18): 5026.0
Pixel (60, 16): 5038.0
Pixel (154, 13): 5042.0
```

```
Pixel (177, 9): 5009.0
Pixel (189, 8): 5014.0
Pixel (189, 12): 5021.0
Pixel (189, 15): 5004.0
Pixel (211, 22): 5019.0
Pixel (225, 37): 5003.0
```

Exemplo 3: Criar uma camada vetorial no formato GeoPackage representando pixels como polígonos.

Neste exemplo, além de percorrer os pixels do raster, o script cria uma camada vetorial temporária, onde cada pixel selecionado é representado por um polígono. Cada polígono inclui um atributo armazenando o valor original do pixel. A camada vetorial é salva em um arquivo GeoPackage.

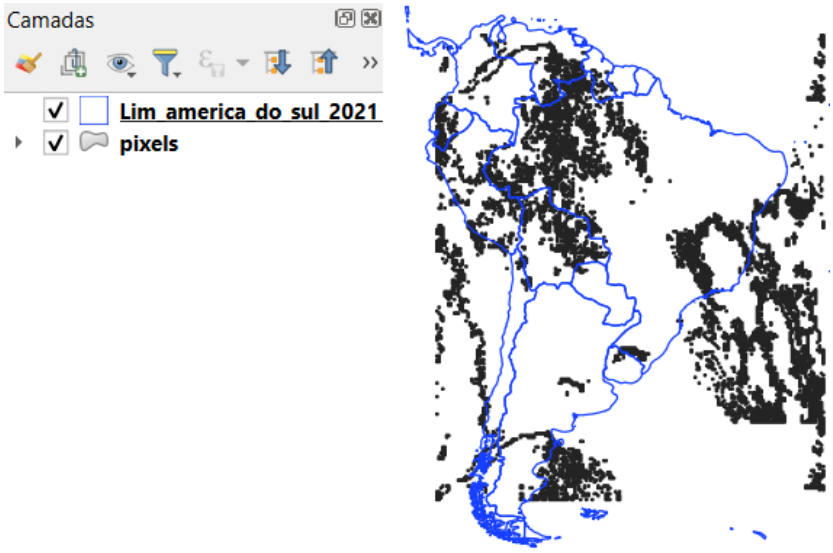
O intervalo de precipitação considerado será de 1000 a 1500 mm.

```
# Caminho do arquivo raster
caminho_raster = 'D:/pasta/trmm/2013_09.tif'
# Caminho do arquivo GeoPackage de saída
caminho_saida = 'D:/pasta/trmm/pixels.gpkg'

# Carrega a camada raster
camada = QgsRasterLayer(caminho_raster, 'imagem')
provedor = camada.dataProvider()
banda = 1  # Banda a ser utilizada

extensao = camada.extent()
resolucao_x = camada.rasterUnitsPerPixelX()
resolucao_y = camada.rasterUnitsPerPixelY()

colunas = int(extensao.width() / resolucao_x)
linhas = int(extensao.height() / resolucao_y)
bloco = provedor.block(banda, extensao, colunas, linhas)

# Cria a camada vetorial do tipo polígono
campos = QgsFields()
campos.append(QgsField('valor', QVariant.Double))

camada_saida = QgsVectorLayer('Polygon?crs=EPSG:4326', 'pixels', 'memory')
provedor_saida = camada_saida.dataProvider()
provedor_saida.addAttributes(campos)
camada_saida.updateFields()

# Percorre os pixels e cria polígonos
for linha in range(linhas):
    for coluna in range(colunas):
        valor = bloco.value(coluna, linha)

        if valor >= 1000 and valor <= 1500:
            # Calcula coordenadas do pixel
```

```
                x_min = extensao.xMinimum() + (coluna * resolucao_x)
                x_max = x_min + resolucao_x
                y_max = extensao.yMaximum() - (linha * resolucao_y)
                y_min = y_max - resolucao_y

                # Cria retângulo representando o pixel
                retangulo = QgsRectangle(x_min, y_min, x_max, y_max)
                geometria = QgsGeometry.fromRect(retangulo)

                # Cria feição
                feicao = QgsFeature()
                feicao.setGeometry(geometria)
                feicao.setAttributes([valor])

                # Adiciona feição
                provedor_saida.addFeature(feicao)

camada_saida.updateExtents()

# Salva a camada temporária em um GeoPackage
transform_context = QgsProject.instance().transformContext()
options = QgsVectorFileWriter.SaveVectorOptions()
options.driverName = 'GPKG'
options.layerName = "pixels"
options.actionOnExistingFile = QgsVectorFileWriter.CreateOrOverwriteFile

erro, mensagem = QgsVectorFileWriter.writeAsVectorFormatV2(
    camada_saida,
    caminho_saida,
    transform_context,
    options
)

if erro == QgsVectorFileWriter.NoError:
    print(f"Arquivo GeoPackage criado com sucesso: {caminho_saida}")
else:
    print(f"Erro ao criar GeoPackage: {mensagem}")
```

Resultado:

O arquivo pixels.gpkg conterá uma camada vetorial com polígonos representando os pixels cuja precipitação acumulada se encontra no intervalo especificado [1000, 1500] mm. Cada polígono terá um atributo valor correspondente à precipitação original registrada no raster.



VIII. Criar e exportar imagens raster com PyQGIS e GDAL

Introdução ao GDAL para criação de imagens

O GDAL (Geospatial Data Abstraction Library) é uma biblioteca fundamental para manipulação de dados geoespaciais, permitindo, entre outras funções, a criação e a gravação de arquivos raster em diversos formatos. A criação de uma imagem com GDAL envolve três passos principais:

1. Definir o driver de saída: especificar o formato do arquivo raster. Por exemplo:

```
driver = gdal.GetDriverByName('GTiff') # Define o driver para salvar o raster
```

2. Criar o dataset: estabelecer as dimensões, o número de bandas e o tipo de dados. Por exemplo:

```
# Cria o dataset de saída
saida_ds = driver.Create(
    caminho_saida, # local onde o arquivo será salvo
    colunas, # quantidade de colunas
    linhas, # quantidade de linhas
    1,  # quantidade de bandas banda
    gdal.GDT_Float32   # tipo de dado do valor de cada pixel
)
```

3. Configurar a georreferência: definir a transformação espacial (geotransform) e o sistema de referência de coordenadas (projeção). Por exemplo:

```
# Abre a imagem de referência para obter informações de geotransformação
ds = gdal.Open(caminho_a)
geotransform = ds.GetGeoTransform()
projecao = ds.GetProjection()
# Define a geotransformação e a projeção
saida_ds.SetGeoTransform(geotransform)
saida_ds.SetProjection(projecao)
```

4. Escrever os dados: transferir a matriz de dados para a nova imagem.

```
saida_ds.GetRasterBand(1).WriteArray(array_diferenca)
```

Exemplo 4: Calcular a diferença entre duas imagens raster e salvar o resultado em uma nova imagem no formato GeoTIFF.

Neste exemplo, será demonstrado como calcular a diferença entre duas imagens raster utilizando o PyQGIS para a leitura e o processamento dos dados, e o GDAL para a criação e gravação de uma nova imagem raster no formato GeoTIFF.

```
# Importa a biblioteca Numpy
import numpy as np

# Caminhos das imagens de entrada
caminho_a = 'D:/bdgeo/raster/trmm/2013_09.tif'
caminho_b = 'D:/bdgeo/raster/trmm/2013_10.tif'
# Caminho de saída
caminho_saida = 'D:/bdgeo/raster/trmm/diferenca2.tif'

# Carrega as imagens raster
raster_a = QgsRasterLayer(caminho_a, 'imagem_a')
raster_b = QgsRasterLayer(caminho_b, 'imagem_b')

# Verifica se as camadas foram carregadas corretamente
if not raster_a.isValid() or not raster_b.isValid():
    print("Erro ao carregar as imagens raster.")
    exit()

# Obtém o provedor de dados
provedor_a = raster_a.dataProvider()
provedor_b = raster_b.dataProvider()

# Considera a banda 1 para ambas
banda = 1

# Obtém a extensão e a resolução
extensao = raster_a.extent()
resolucao_x = raster_a.rasterUnitsPerPixelX()
resolucao_y = raster_a.rasterUnitsPerPixelY()

# Calcula número de colunas e linhas
colunas = int(extensao.width() / resolucao_x)
linhas = int(extensao.height() / resolucao_y)

# Obtém blocos de dados
bloco_a = provedor_a.block(banda, extensao, colunas, linhas)
bloco_b = provedor_b.block(banda, extensao, colunas, linhas)
```

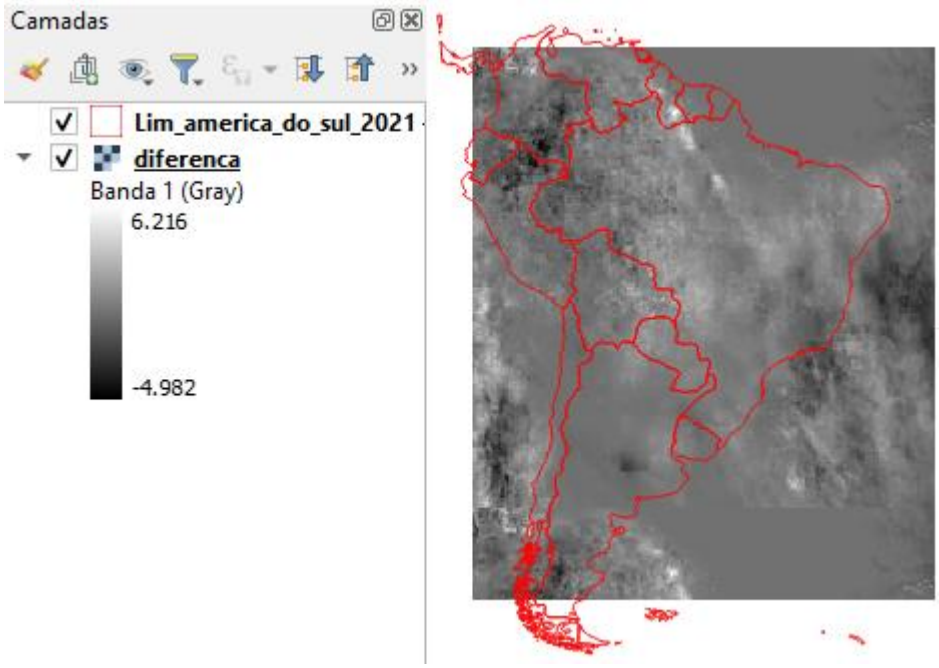
```
# Cria uma matriz para armazenar a diferença
array_diferenca = np.zeros((linhas, colunas), dtype=np.float32)

# Percorre os pixels e calcula a diferença
for linha in range(linhas):
    for coluna in range(colunas):
        valor_a = bloco_a.value(coluna, linha)
        valor_b = bloco_b.value(coluna, linha)
        diferenca = valor_b - valor_a
        array_diferenca[linha, coluna] = diferenca

# --- Salvando a imagem de diferença ---

# Abre a imagem de referência para obter informações de geotransformação
ds = gdal.Open(caminho_a)
geotransform = ds.GetGeoTransform()
projecao = ds.GetProjection()

# Define o driver para salvar o raster
driver = gdal.GetDriverByName('GTiff')

# Cria o dataset de saída
saida_ds = driver.Create(
    caminho_saida,
    colunas,
    linhas,
    1,  # uma banda
    gdal.GDT_Float32
)

# Define a geotransformação e a projeção
saida_ds.SetGeoTransform(geotransform)
saida_ds.SetProjection(projecao)

# Escreve os dados no raster de saída
saida_ds.GetRasterBand(1).WriteArray(array_diferenca)

# Define o valor NoData (opcional)
saida_ds.GetRasterBand(1).SetNoDataValue(-9999)

# Salva e fecha
saida_ds.FlushCache()
saida_ds = None

print(f"Imagem de diferença salva com sucesso em: {caminho_saida}")
```

Descrição do Programa:

1. Entrada: Caminhos das duas imagens raster que se deseja comparar;

2. Processamento:
    - Leitura das duas imagens;
    - Cálculo da diferença entre os valores correspondentes de cada pixel (valor_b - valor_a);
    - Armazenamento do resultado em uma matriz NumPy.
3. Saída: Criação de uma nova imagem GeoTIFF com os valores de diferença;
4. Tecnologias utilizadas:
    - PyQGIS para ler os valores dos pixels;
    - GDAL para gravar a nova imagem raster;
    - NumPy para armazenar e manipular os dados numéricos.

Resultado do programa:



Exercícios

Veja os vídeos se tiver dúvidas nos exercícios:

Exercício 1: https://youtu.be/Lbd0wXTji3A

Exercício 2: https://youtu.be/q2YtKAC7eyE

Exercício 3 - https://youtu.be/GqlbN9mVwQ0

Exercício 4 - https://youtu.be/knVpZC-GA4E

Exercício 5 - https://youtu.be/0jm-438OhYs

Descrição da imagem: Para realizar os exercícios, utilize a imagem landsat9OLI_2023.tif, composta por 6 bandas. Esta imagem representa a média anual dos valores de reflectância dos pixels obtidos pelo sensor OLI (Operational Land Imager) a bordo do satélite Landsat 9, referentes ao ano de 2023.

A imagem foi gerada a partir de processamento no Google Earth Engine (GEE), que permitiu compilar e sintetizar diversas cenas do ano, reduzindo a presença de nuvens e ruídos, resultando em uma composição limpa e representativa do comportamento espectral médio da região.

Composição das bandas:

1. B1 (Coastal/Aerosol) – 0.43-0.45 µm: detecção de partículas finas (aerossóis), qualidade atmosférica e correção de efeitos atmosféricos;
2. B2 (Blue) – 0.45-0.51 µm: análise de corpos d'água, vegetação e saúde das plantas;
3. B3 (Green) – 0.53-0.59 µm: monitoramento da vegetação e uso em composições coloridas naturais;
4. B4 (Red) – 0.64-0.67 µm: diferenciação entre vegetação e solo, análise da cobertura terrestre;
5. B5 (Near Infrared - NIR) – 0.85-0.88 µm: identificação de vegetação saudável e análise de umidade;
6. B6 (Shortwave Infrared 1 - SWIR 1) – 1.57-1.65 µm: mapeamento de umidade no solo e diferenciação de tipos de solo e vegetação;

Exercício 1: Escreva um script em PyQGIS para listar:

- O número total de bandas da imagem;
- O intervalo de valores mínimo e máximo de cada banda.

Algoritmo sugerido para desenvolver o script:

1. Carregar a imagem raster usando `QgsRasterLayer`;
2. Verificar se a camada foi carregada com sucesso;
3. Obter o provedor de dados da banda;
4. Obter o número total de bandas usando o método `bandCount()` do provedor de dados da banda;
5. Para cada banda:
   a. Usar o método `.bandStatistics()`, do provedor de dados da banda, para obter as estatísticas;
   b. Extrair o valor mínimo (minimumValue) e máximo (maximumValue) da banda.
6. Exibir os resultados com `print()`.

Resultado esperado:

```
Número total de bandas: 19
Banda 1: Mínimo = -0.18839499354362488, Máximo = 0.5292999744415283
Banda 2: Mínimo = -0.16994249820709229, Máximo = 0.5685700178146362
Banda 3: Mínimo = -0.0635175034403801, Máximo = 0.5996724963188171
Banda 4: Mínimo = -0.09000000357627869, Máximo = 0.6438375115394592
Banda 5: Mínimo = -0.017977500334382057, Máximo = 0.7165200114250183
Banda 6: Mínimo = -0.0028250000905245543, Máximo = 0.8835824728012085
```

Exercício 2: Extração de banda específica de imagem raster.

O código a seguir copia a Banda 5 de uma imagem Landsat 9 OLI para o arquivo banda5.tif.

Sua tarefa: Altere o script para copiar a Banda 3 da imagem para um novo arquivo chamado banda3.tif.

Código base:

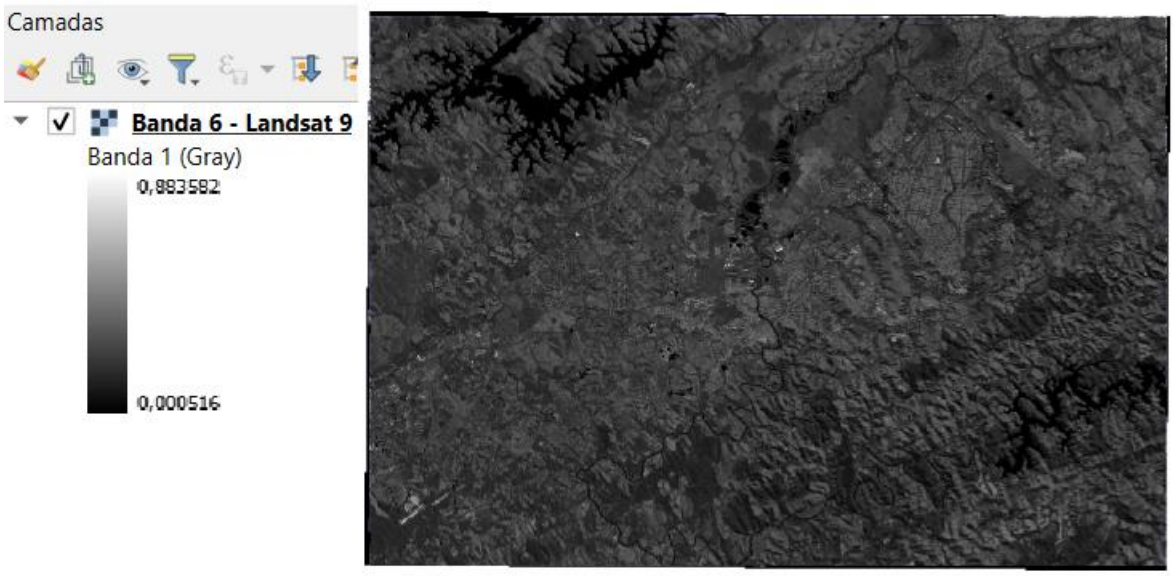
```
import numpy as np

# Caminho da imagem de entrada
caminho_entrada = 'D:/pasta/landsat9OLI_2023.tif'

# Caminho do arquivo de saída
caminho_saida = 'D:/pasta/banda5.tif'

numero_banda = 5

# 1. Carrega a imagem Landsat (todas as bandas)
landsat = QgsRasterLayer(caminho_entrada, 'Landsat9')
if not landsat.isValid():
    raise Exception("Erro ao carregar imagem Landsat!")

# 2. Verifica se a banda existe
if numero_banda > landsat.bandCount():
    raise Exception(f"A imagem só tem {landsat.bandCount()} bandas!")

# 3. Configura o cálculo para extrair apenas a banda 5
entries = []
band_entry = QgsRasterCalculatorEntry()
band_entry.ref = f'banda{numero_banda}@{1}'  # Referência única
band_entry.raster = landsat
band_entry.bandNumber = numero_banda
entries.append(band_entry)

# Fórmula simples que mantém o valor original
formula = f'banda{numero_banda}@{1}'

# 4. Executa a extração
extracao = QgsRasterCalculator(
    formula,
    caminho_saida,
    'GTiff',  # Formato GeoTIFF
    landsat.extent(),
    landsat.width(),
    landsat.height(),
    entries
)

resultado = extracao.processCalculation()

# 5. Verifica e carrega o resultado
if resultado == 0:  # Código 0 = sucesso
```

```
        banda = QgsRasterLayer(caminho_saida, f'Banda {numero_banda} - Landsat 9')
        QgsProject.instance().addMapLayer(banda)
        print(f"Banda {numero_banda} extraída com sucesso para: {caminho_saida}")
    else:
        raise Exception("Erro ao extrair a banda!")
```

Orientações para a alteração:

- Atualize a variável numero_banda para 6;
- Atualize a variável caminho_saida para 'D:/pasta/banda6.tif';
- Mantenha a lógica de verificação e extração.

Resultado esperado:



Exercício 3: O código a seguir calcula o NDVI (Índice de Vegetação por Diferença Normalizada) e salva o resultado no arquivo NDVI.tif.

Sua tarefa: altere este código para calcular o NDWI (Índice de Água por Diferença Normalizada) e salve o resultado no arquivo NDWI.tif.

Fórmula do NDVI:

```
NDVI = (NIR - RED)/(NIR + RED)
```

Para o sensor OLI Landsat 9, as bandas relevantes são:

- Banda 5 (NIR) – Infravermelho próximo
- Banda 4 (RED) – Vermelho

Fórmula do NDWI:

```
NDWI = (Green - NIR)/(Green + NIR)
```

Bandas relevantes:

- Banda 3 (Green) – Verde
- Banda 5 (NIR) – Infravermelho próximo

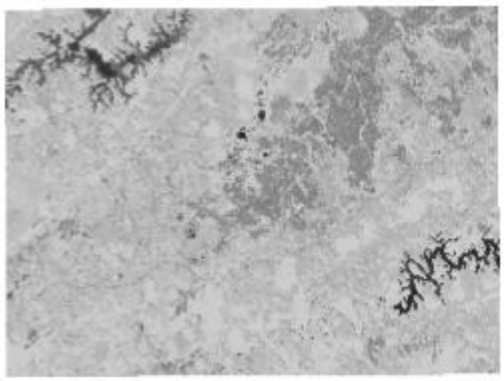
Código base:

```
from osgeo import gdal, gdal_array
import numpy as np
```

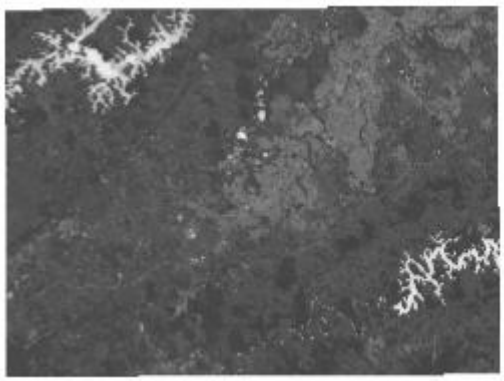
```
# Configurações de entrada e saída
caminho_entrada = 'D:/pasta/landsat9OLI_2023.tif'
caminho_saida = 'D:/pasta/NDVI.tif'

# Bandas Landsat 9 OLI
BANDA_NIR = 5  # Infravermelho Próximo
BANDA_RED = 4  # Vermelho

# Carrega as bandas diretamente com GDAL
dataset = gdal.Open(caminho_entrada)
nir = dataset.GetRasterBand(BANDA_NIR).ReadAsArray().astype(float)
red = dataset.GetRasterBand(BANDA_RED).ReadAsArray().astype(float)

# Cálculo do NDVI com tratamento de divisão por zero
ndvi = np.divide(
    (nir - red), # numerador da divisão
    (nir + red), # denominador da divisão
    out=np.full_like(nir, -1, dtype=float),  # Valor padrão para divisão por zero
    where=(nir + red) != 0
)

# Salva o resultado
driver = gdal.GetDriverByName('GTiff')
output = driver.Create(
    caminho_saida,
    dataset.RasterXSize,
    dataset.RasterYSize,
    1,
    gdal.GDT_Float32
)

output.GetRasterBand(1).WriteArray(ndvi)
output.SetGeoTransform(dataset.GetGeoTransform())
output.SetProjection(dataset.GetProjection())
output.GetRasterBand(1).SetNoDataValue(-1)
output.FlushCache()

print("NDVI calculado!")
```

Resultado esperado:

NDVI



NDWI



Explicação de partes do código:

Carregar as bandas com GDAL (Geospatial Data Abstraction Library):

1. `dataset = gdal.Open(caminho_entrada)`

- **gdal.Open()**: função que **abre** o arquivo raster especificado no caminho `caminho_entrada`;
- O arquivo pode ser, por exemplo, um .tif (GeoTIFF);
- Retorna um objeto **Dataset**, que representa o arquivo raster e fornece acesso às suas **bandas**, **propriedades espaciais** e **metadados**.

2. `dataset.GetRasterBand(BANDA_NIR)`

- Método que **acessa uma banda específica** do raster;
- As bandas são numeradas a partir de **1** (não a partir de zero, como é comum em Python);
- O método retorna um objeto **Band**, que permite **ler, escrever** e consultar informações sobre a banda.

3. `ReadAsArray()`

- Método que **lê os valores da banda** como uma **matriz NumPy**;
- Cada elemento da matriz representa o **valor de reflectância** (ou outro valor) de um pixel da imagem;
- Permite trabalhar com os dados raster em operações vetorizadas, como cálculos matemáticos.

4. `.astype(float)`

- Converte os valores da matriz para o tipo **float**;
- Essa conversão é essencial para operações aritméticas que podem resultar em **valores fracionários**, como índices espectrais (NDVI, NDWI etc);
- Além disso, evita erros causados por operações com tipos inteiros, como **divisão truncada**;

Divisão usando Numpy:

1. `np.divide(a, b, out=..., where=...)`

- `np.divide()`: função do NumPy que realiza a divisão **elemento a elemento** (pixel a pixel) entre duas matrizes, com tratamento opcional para evitar erros como divisão por zero;
- Aqui está dividindo:

  Numerador: `(nir - red)`

Denominador: `(nir + red)`

2. `out=np.full_like(nir, -1, dtype=float)`

- **out=**: define onde será armazenado o resultado da divisão;
- **np.full_like(nir, -1, dtype=float)**: cria uma **nova matriz**, com o **mesmo formato** de nir, preenchida com o valor **-1** e com tipo de dado float;
- O valor **-1** será usado como **valor padrão** (ou **NoData**) para os casos em que a divisão não puder ser realizada, como em divisões por zero.

3. `where=(nir + red) != 0`

- Define a **condição** para que a divisão seja feita **apenas onde o denominador for diferente de zero**;
- Se (nir + red) == 0, a divisão é **evitada** e o resultado correspondente permanece como **-1** (o valor padrão);
- Evita o erro de **divisão por zero** e o surgimento de valores inf (infinito) ou nan (não numérico).

Resumo lógico:

- Para cada pixel:
  - Se nir + red ≠ 0 → calcula: (nir - red) / (nir + red)
  - Se nir + red = 0 → coloca -1 no resultado.

Exercício 4: Complete o código a seguir para recortar a imagem NDWI.tif (gerada no exercício anterior), utilizando como máscara os limites do município de Jacareí. O resultado do recorte deve ser salvo em um novo arquivo chamado recorte.tif.

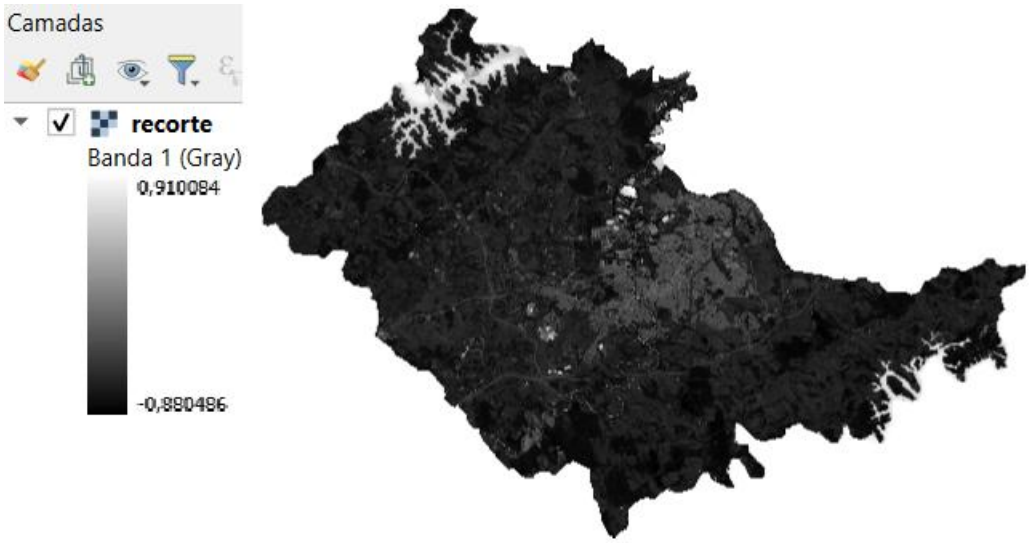
Código base:

```
caminho_mun = 'D:/pasta/municipio.gpkg'
caminho_raster = 'D:/pasta/NDWI.tif'
caminho_saida = 'D:/pasta/recorte.tif'

# Carrega as camadas
camada_mun = QgsVectorLayer(caminho_mun, 'mun', 'ogr')
camada_raster = QgsRasterLayer(caminho_raster, 'ndwi')

# Cria um QgsFeatureRequest para filtrar apenas o município de Jacareí
request = QgsFeatureRequest().setFilterExpression("municipio = 'Jacareí'")

# Obtém a feição correspondente ao município de Jacareí
feicao_jacarei = next(camada_mun.getFeatures(request))

# Cria uma camada temporária contendo apenas a feição de Jacareí
camada_jcr = QgsVectorLayer("Polygon?crs=EPSG:4326", "jacarei", "memory")

# Adiciona a feição à camada temporária
camada_jcr.dataProvider().addFeature(feicao_jacarei)
```

Orientações adicionais:

- Utilize o algoritmo processing.run("gdal:cliprasterbymasklayer", ...) ou equivalente, para realizar o recorte da imagem raster, utilizando a camada camada_jcr como máscara;
- Certifique-se de que as camadas raster e vetorial estejam no mesmo sistema de referência de coordenadas (CRS);
- O resultado do recorte deve ser salvo no caminho especificado em caminho_saida.

Resultado esperado:



Exercício 5: Complete o código a seguir para calcular, para cada banda da imagem landsat9OLI_2023.tif, as seguintes estatísticas na área correspondente ao município de Jacareí:

- a quantidade de pixels válidos (count);
- o valor médio (mean);
- a soma dos valores dos pixels (sum).

O resultado de cada estatística deve ser exibido no console e, adicionalmente, armazenado como atributo na feição da camada de saída, que está na variável camada_jcr.

Código base:

```
caminho_mun = 'D:/pasta/municipio.gpkg'
caminho_raster = 'D:/pasta/landsat9OLI_2023.tif'

# Carrega as camadas
camada_mun = QgsVectorLayer(caminho_mun, 'mun', 'ogr')
camada_raster = QgsRasterLayer(caminho_raster, 'ndwi')

# Cria um QgsFeatureRequest para filtrar apenas Jacareí
request = QgsFeatureRequest().setFilterExpression("municipio = 'Jacareí'")
```

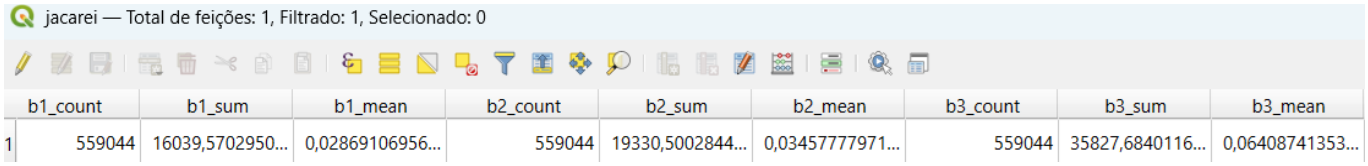
```
# Obtém feição de Jacareí
feicao_jacarei = next(camada_mun.getFeatures(request))

# Cria uma camada temporária em memória com apenas Jacareí
camada_jcr = QgsVectorLayer("Polygon?crs=EPSG:4326", "jacarei", "memory")
# Adiciona a feição na camada temporária
camada_jcr.dataProvider().addFeature(feicao_jacarei)
camada_jcr.updateExtents()

# Adiciona a camada ao projeto (opcional, para visualização)
QgsProject.instance().addMapLayer(camada_jcr)

# Número de bandas no raster
num_bandas = camada_raster.bandCount()

print(f"Número de bandas: {num_bandas}")
```

Orientações adicionais:

- Utilize o método QgsZonalStatistics ou o algoritmo qgis:zonalstatisticsfb para calcular as estatísticas;
- Os atributos resultantes devem ser adicionados à feição camada_jcr com um prefixo indicativo da banda;
- Para cada banda, exiba os valores de count, sum e mean no console.

Resultado esperado:

Parte da tabela de atributos da camada jacarei:



	b1_count	b1_sum	b1_mean	b2_count	b2_sum	b2_mean	b3_count	b3_sum	b3_mean
1	559044	16039,5702950...	0,02869106956...	559044	19330,5002844...	0,03457777971...	559044	35827,6840116...	0,06408741353...

Resultado exibido no console:

```
Número de bandas: 6
Banda 1:
 - Contagem: 559044.0
 - Soma: 16039.57029501029
 - Média: 0.028691069566993455
Banda 2:
 - Contagem: 559044.0
 - Soma: 19330.50028441674
 - Média: 0.034577779717547705
Banda 3:
 - Contagem: 559044.0
 - Soma: 35827.68401163953
 - Média: 0.06408741353388916
Banda 4:
 - Contagem: 559044.0
 - Soma: 32159.344132465543
```

```
 - Média: 0.057525604661646565
Banda 5:
 - Contagem: 559044.0
 - Soma: 162747.31819102686
 - Média: 0.2911171896863697
Banda 6:
 - Contagem: 559044.0
 - Soma: 108966.578640266
 - Média: 0.19491592547324718
```