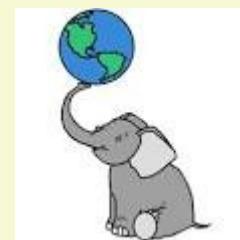


Apostila Python + QGIS



André Luiz Lima Costa

Janeiro 2020

Sumário

1. QGIS.....	7
1.0 Introdução.....	7
1.1 Instalando e Iniciando o QGIS.....	7
1.2 Carregando dados Vetoriais.....	8
Dados no formato texto.....	8
Dados no Formato GIS (ESRI Shapefile, GMT, MapInfo, etc).....	10
Abrindo Dados de Fontes Remotas (Banco de dados em servidores remotos).....	11
Propriedades e atributos dos objetos espaciais.....	14
1.3 Carregando dados Raster.....	18
1.4 Criando Dados Vetoriais.....	24
1.5 Criando Raster a partir de Pontos Vetoriais.....	28
2. Fundamentos de Python.....	31
2.0 A linguagem Python – Introdução.....	31
Instalação.....	31
Console Python.....	31
2.1 Fundamentos da linguagem Python.....	32
Tipos Numéricos.....	32
Tipo Alfanumérico.....	32
Tipo lista.....	33
Tuples.....	36
Sets.....	36
Dicionários.....	36
2.2 Controles de fluxo.....	38
if elif else.....	38
for.....	39
while.....	39
range().....	39
Break, continue e else em loops.....	40
2.3 Funções.....	41
2.4 Módulos.....	43

2.5 Pandas.....	43
2.6 Gráficos.....	46
3. Usos do Python no QGIS.....	48
3.1 Primeiros passos, noções de Classes.....	48
Classe Projeto (QgsProject) – Criar e ler um Projeto.....	48
Classe Vetor (QgsVectorLayer) – Adicionar Camada Vetorial.....	48
Classe Raster (QgsRasterLayer) – Adicionar Camada Raster.....	51
3.2 Interagindo com informações de objetos da classe Vector.....	52
O sistema de referência de coordenadas CRS (Coordinate Reference System).....	53
A extensão da Camada.....	53
Quantidade de itens.....	53
Obtendo informações dos campos de atributos.....	54
Metadata de camada vetorial.....	54
Obtendo os elementos de cada item da camada vetorial.....	58
3.3 Interagindo com informações de objetos da classe Raster.....	58
Informações de dimensão do objeto Raster.....	59
Raster com uma banda de valores.....	60
Raster com mais de uma banda de valores.....	62
3.4 Criando objeto vetorial.....	64
Ponto.....	64
Linha.....	67
Polígono.....	68
3.5 Criando objeto raster.....	69
4. Executando python scripts fora do Qgis.....	72
5. Integração PostgreSQL/Postgis com PyQgis.....	74
5.1 O Postgis.....	74
5.2 Instalando PostgreSQL/Postgis.....	75
5.2.1 - Windows.....	75
5.2.2 - Linux - Centos ou RedHat (PostgreSQL e Postgis).....	79
5.2.3 - OSX.....	79
5.3 Criando um banco de dados Postgis.....	80
5.4 Entrando dados no postgis.....	80
5.4.1 Conectando com o banco de dados.....	80

5.4.2 Inserindo dados vetoriais no banco de dados.....	81
Pontos.....	81
Linhas.....	82
Poligonos.....	83
5.4.3 Inserindo dados raster no banco de dados.....	84
5.5 Visualizando os dados criados.....	85
5.6 Extrairando dados do Postgis usando pyQgis.....	86
5.6.1 Conexão ao banco de dados e tabelas usadas.....	86
5.6.2 Obtendo valores distintos de uma coluna da tabela.....	89
5.6.3 Usando a cláusula WHERE.....	89
5.6.4 Selecionando dados espaciais e criando arquivo do resultado.....	91
6. Introdução à análise espacial no pyQgis.....	92
6.1 O conceito de análise espacial.....	92
6.1.1 Exemplo 1 – A primeira análise espacial.....	92
Preparando.....	93
Executando.....	93
O resultado da análise.....	98
6.1.2 Exemplo 1 – Um outro exemplo usando somente pyQgis.....	98
Preparando.....	98
Executando.....	99
O resultado da análise.....	99
6.1.3 Considerações sobre as duas análises espaciais básicas efetuadas.....	100
7. Análise espacial em dados vetoriais.....	101
7.1 Geopandas crash course.....	101
7.1.1 Instalação.....	101
7.1.2 O básico.....	101
Entrando com os dados.....	101
Obtendo informações do objeto.....	101
Dados do Postgis.....	102
Filtrando dados (query).....	102
Gravando objeto em arquivo.....	102
Plotando o objeto.....	102
7.1.3 Relações espaciais de geometrias com o Geopandas.....	103
Checando a posição relativa.....	103
Retornando uma nova geometria.....	104
Métodos construtivos.....	105
7.1.4 Como ficariam o dois exemplos da sessão 1 usando Geopandas.....	105

Casos de cólera.....	105
Festa em Melbourne.....	106
7.2 Case Study – Análise espacial com dados de exploração mineral no Tocantins.....	107
7.2.1 Os dados vetoriais que serão usados.....	107
Carregando os dados.....	107
Conhecendo os dados.....	107
7.2.2 A preparação dos dados.....	108
Filtrando os dados.....	108
Visualizando os dados filtrados.....	109
7.2.3 A análise dos dados.....	109
Por dados de ocorrência mineral.....	109
Por requerimento e concessão de lavra.....	110
Unindo os resultados.....	110
Expandindo o mapa para as unidades geológicas iguais.....	110
7.2.4 Validando o resultado.....	111
7.2.5 Resultado Final.....	112
8. Análise espacial em objeto raster.....	114
8.1 rasterio.....	114
8.1.1 O básico.....	114
Carregando uma imagem raster.....	114
Obtendo informações sobre o objeto raster.....	114
8.1.2 Carregando e plotando uma banda da imagem.....	115
8.1.3 Composição colorida RGB.....	116
8.1.4 Operações entre bandas.....	116
8.2 GDAL.....	117
8.2.1 Conceitos básicos do GDAL.....	118
8.2.2 Usando GDAL via chamada de sistema do python.....	120

Esta apostila é uma compilação de exemplos da web sobre PyQgis e de conhecimento do autor sobre o assunto organizado progressivamente.

Licença:

https://creativecommons.org/licenses/by/4.0/deed.pt_BR.

Atribuição 4.0 Internacional (CC BY 4.0)

Os arquivos de dados usados nessa apostila, bem como esta apostila são encontrados em:

<http://amazeone.com.br/pyqgis/>

VISITE:

amazeone.com.br

1. QGIS

1.0 Introdução

O QGIS é um programa que foi iniciado em 2002 por Gary Sherman e se tornou em um projeto incubador da *Open Source Geospatial Foundation* em 2007. Sua Versão 1.0 foi lançada em Janeiro de 2009.

As bases para o QGIS foram as bibliotecas (libraries) QT, GEOS, OGR/GDAL e GRASS. Usado com o apoio de PostgreSQL-Postgis o QGIS se transforma em uma ferramenta completa para o geoprocessamento e análise espacial de dados.

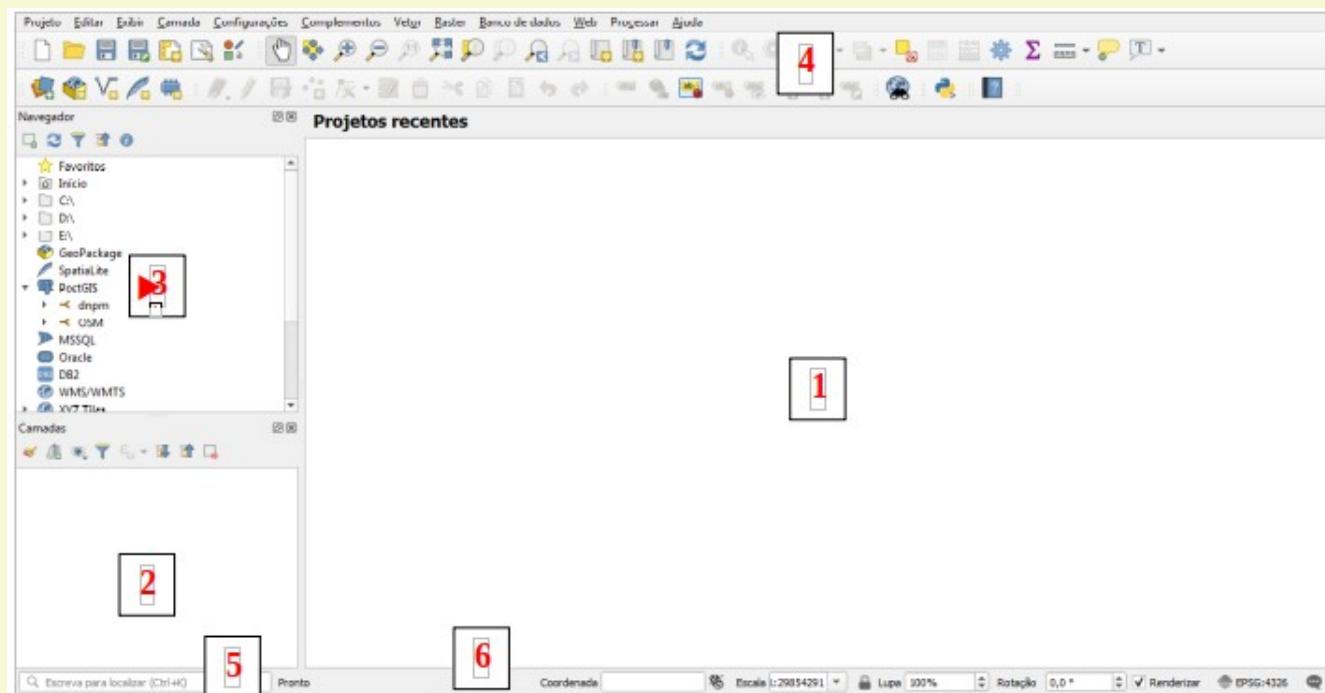
A versão que trabalharemos é a 3.4 LTR (Madeira) que é versão estável mais recente. Versões mais recentes de desenvolvimento já lançadas são a 3.8 (Zanzibar) e 3.10 (A Coruña). O QGIS pode ser instalado em qualquer sistema operacional (Linux, Unix, OSX, Windows, Android) e já possui o python dentro dele. O python script é a ferramenta principal de interação para tarefas mais complexas ou repetitivas e é usado também para o desenvolvimento dos *plugins*.

Veremos nesse curso a integração da linguagem Python com o QGIS com o objetivo de automatização de processos e análises espaciais de dados bem como a integração com banco de dados geoespaciais. Mas antes vamos falar um pouco sobre o QGIS.

1.1 Instalando e Iniciando o QGIS

O QGIS pode ser instalado em diversos sistemas operacionais. O link abaixo fornece detalhadamente as informações necessárias para a instalação em todos os sistemas operacionais.

https://www.qgis.org/pt_BR/site/forusers/alldownloads.html. Ao iniciar o QGIS veremos a seguinte imagem.



Elementos do Programa:

- 1 - Painel Principal do Mapa – Aqui é onde o mapa é exibido a medida que as camadas são carregadas. Você pode interagir com as camadas carregadas tipo: dar zoom, mover o mapa, selecionar elementos e várias outras operações que veremos adiante.
- 2 - Lista de Camadas Carregadas – A medida que as camadas são carregadas uma lista delas será criada nesse painel, Aqui podemos ativar/desativar a visualização, ordenar, e modificar a aparência das camadas.
- 3 - Navegador – No navegador podemos acessar diversos formatos de dados compatíveis localizados no seu computador, em provedores de dados, em banco de dados, etc.
- 4 - Barra de Ferramentas e Menus – Aqui, como em todos programas, estão os controles do aplicativo divididos nas categorias correspondentes.
- 5 - Pesquisa – Podemos nesse campo acessar/pesquisar rapidamente as ferramentas, controles e processos do QGIS entrando com o nome a ser pesquisado.
- 6 - Barra de Status – Informações gerais sobre projeção, coordenadas do mapa na posição do cursor, escala, rotação e etc. podem ser vistas de forma rápida aqui.

1.2 Carregando dados Vetoriais

Dados vetoriais são informações de determinada(s) grandeza(s) ou descrição, também conhecido como atributos, com uma peculiar distribuição espacial, seja ela do tipo ponto, linha ou polígono.

QGIS pode abrir dados vetoriais de diversos formatos graças à interação com a biblioteca GDAL. Vamos aqui abrir diversos formatos como exemplo.

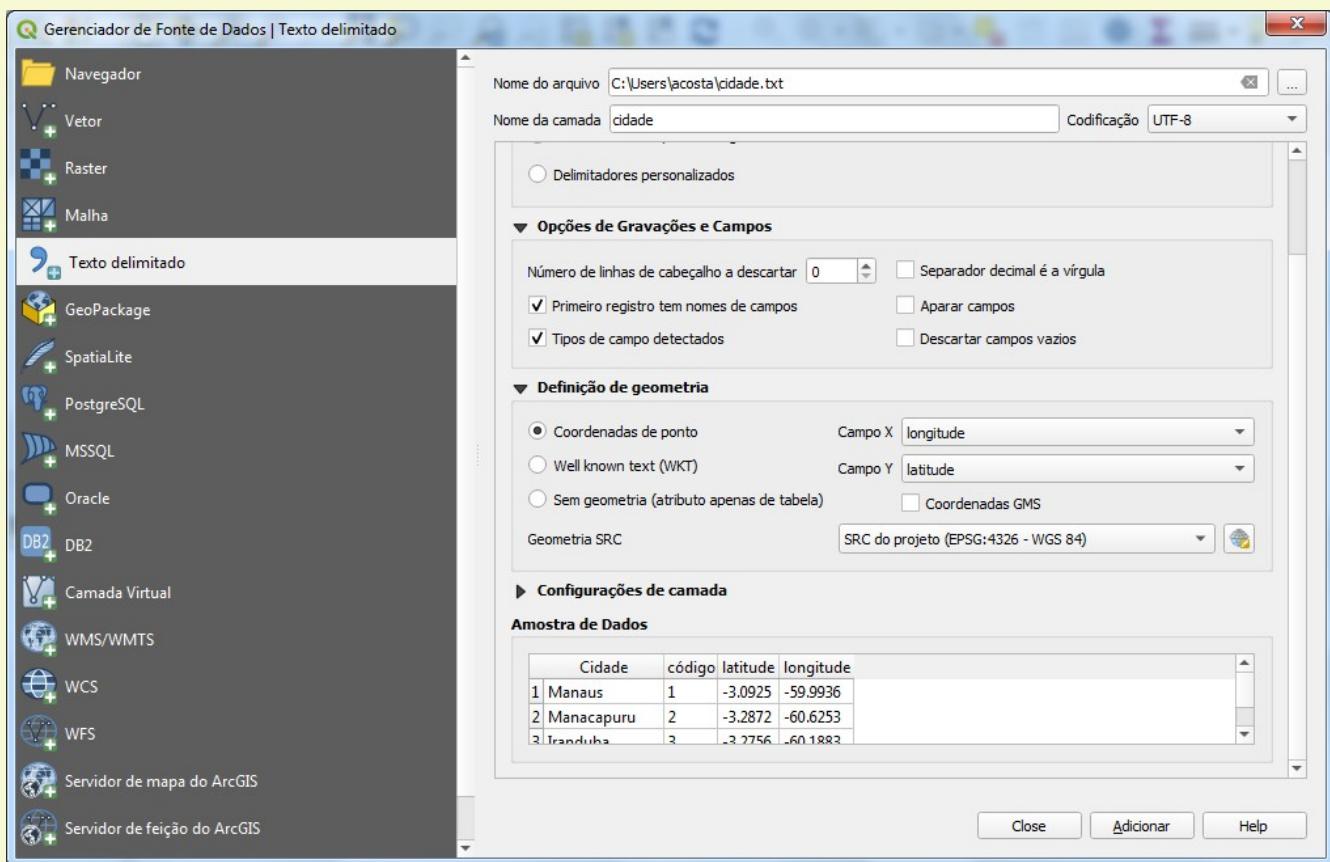
Dados no formato texto

Crie o seguinte arquivo texto e grave o arquivo como ***cidade.txt***.

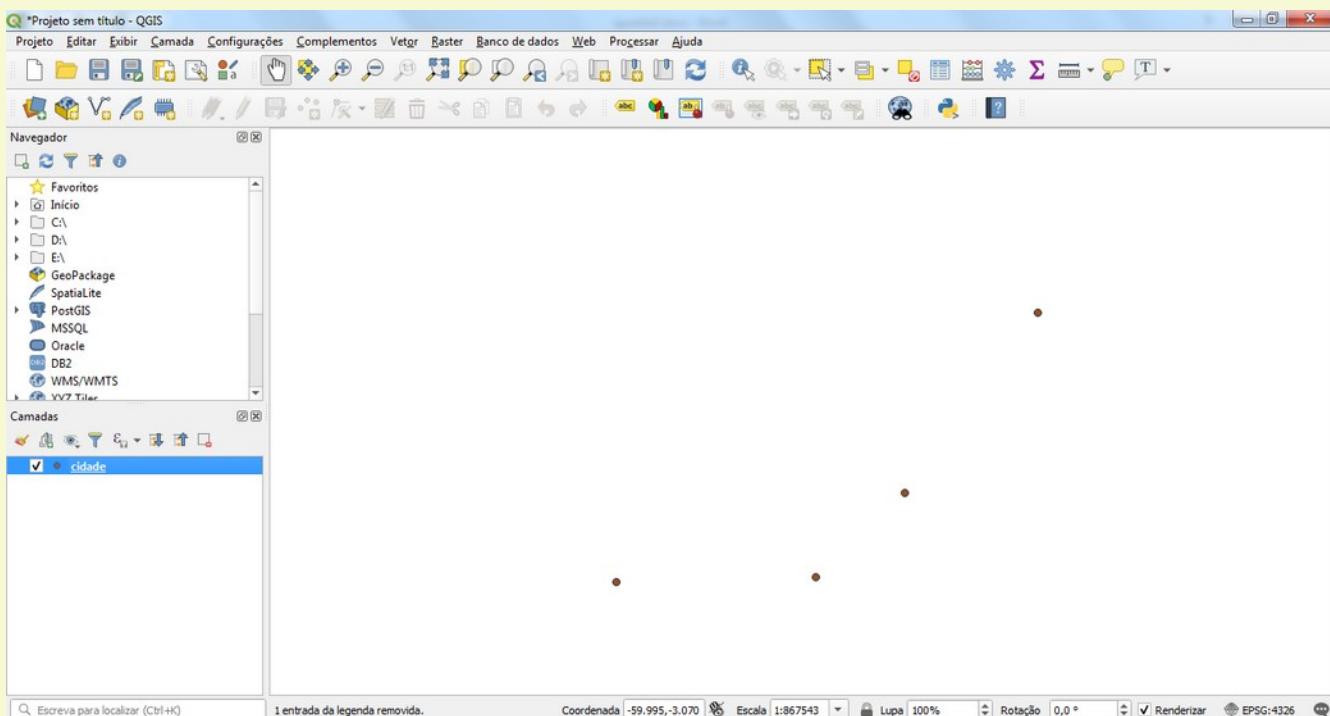
```
Cidade,código,latitude,longitude
Manaus,1,-3.0925,-59.9936
Manacapuru,2,-3.2872,-60.6253
Iranduba,3,-3.2756,-60.1883
Rio Preto de Eva,4,-2.6968,-59.7014
```

- Inicie o QGIS e vá no menu **Camada > Adicionar Camada > Vetorial**.
- Selecione no lado direito a opção Texto Delimitado.
- Navegue até o local do arquivo criado acima no ... e selecione o arquivo ***cidade.txt***.
- Aceite os valores já definidos mas na seção **Definição de Geometria** selecione **SRC do Projeto EPSG:4326 WGS-84**

Todos os campos devem ficar conforme a imagem abaixo e em seguida clique em **Adicionar** e depois em **Close**.

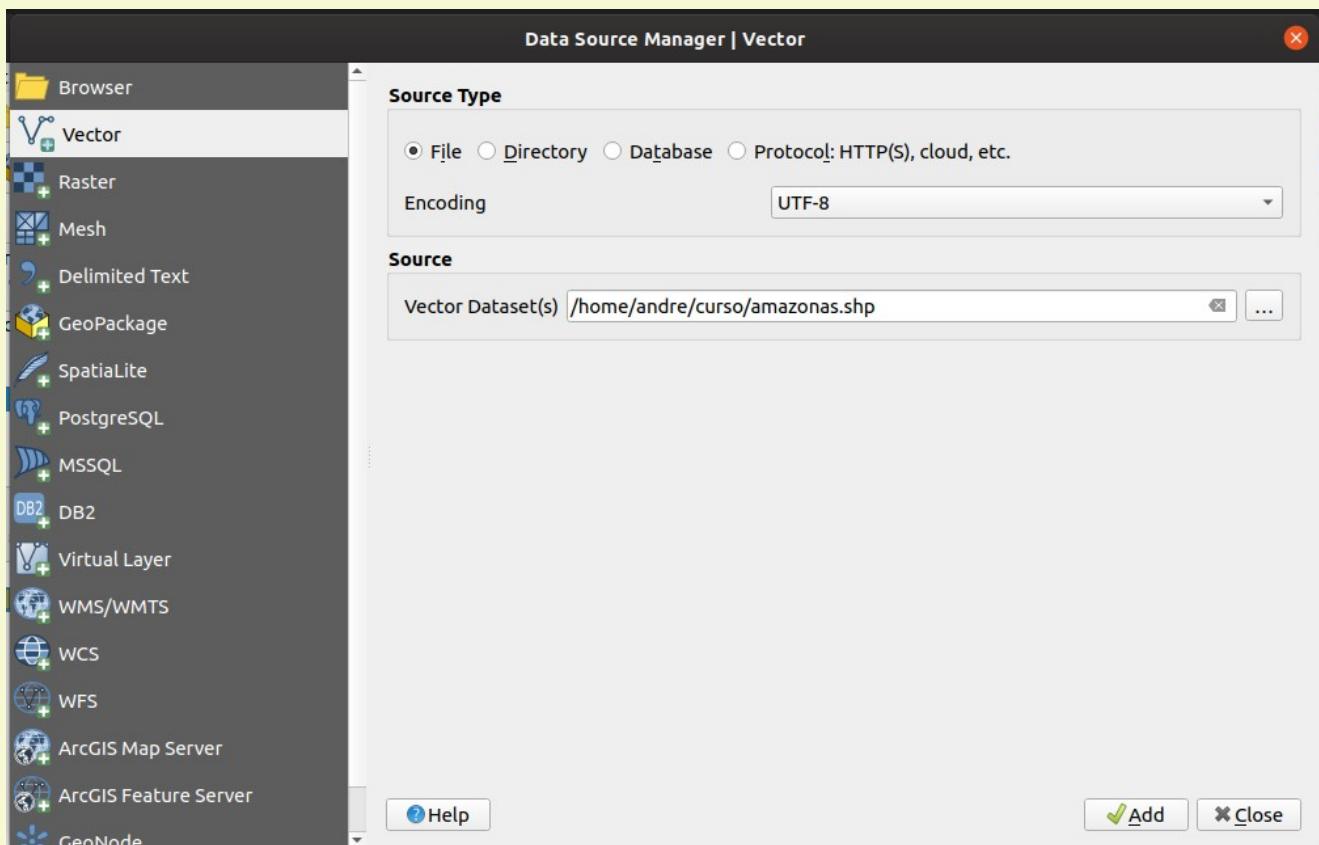


A camada cidade será carregada.

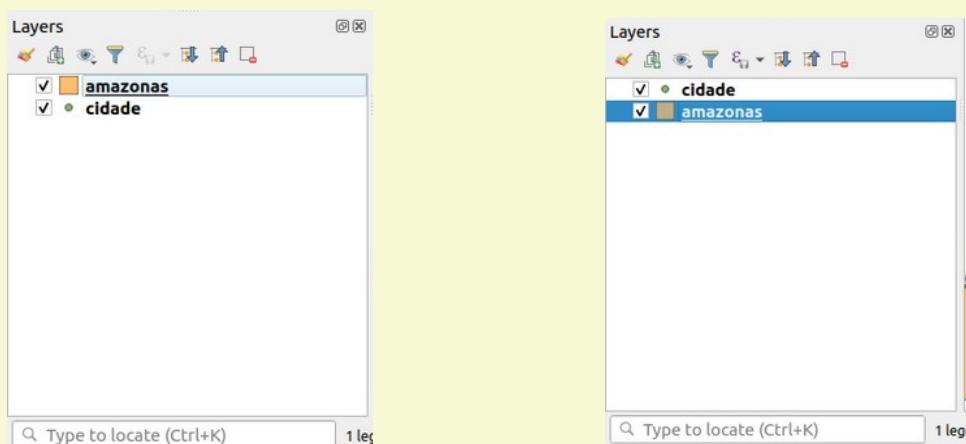


Dados no Formato GIS (ESRI Shapefile, GMT, MapInfo, etc)

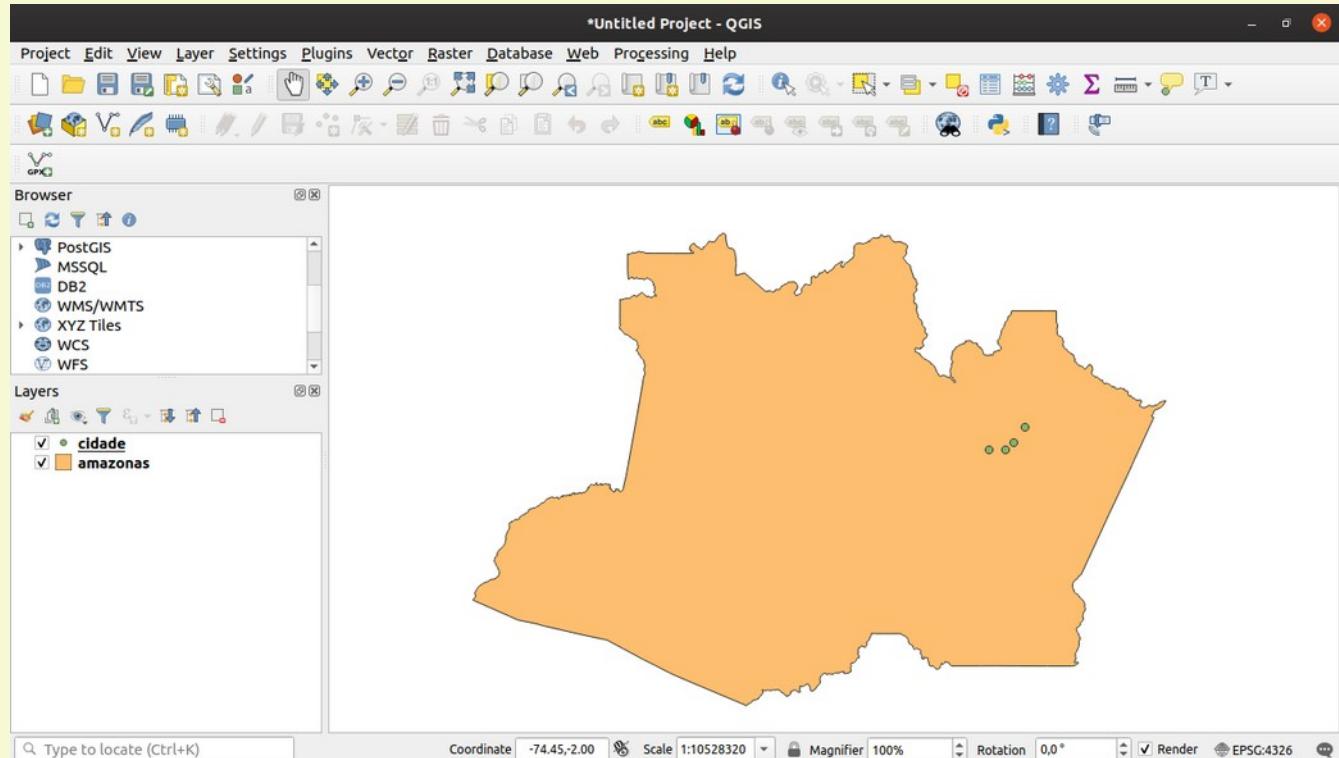
- Vá no menu **Camada > Adicionar Camada > Vetorial**.
- Selecione no lado direito a opção Vector.
- Navegue até o local do arquivo criado acima no botão ... e selecione o arquivo **amazonas.shp**.
Todos os campos devem ficar conforme a imagem abaixo e em seguida clique em **Adicionar** e depois em **Close**.



Posicione a camada recém adicionada abaixo da camada cidade para ficar como a imagem de direita.
Faça isso clicando e arrastando na camada amazonas.



Abaixo vemos o resultado após usarmos o zoom -.

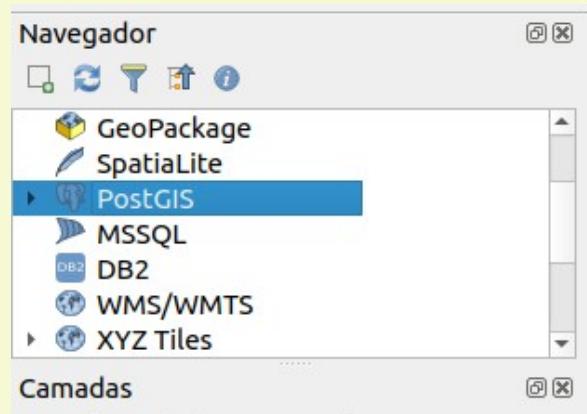


Podemos gravar o nosso projeto com o nome de primeiro. Vá até o menu **Projeto > Salvar**. Na janela que aparece escreva o nome '**primeiro**'. Pronto, o projeto está salvo.

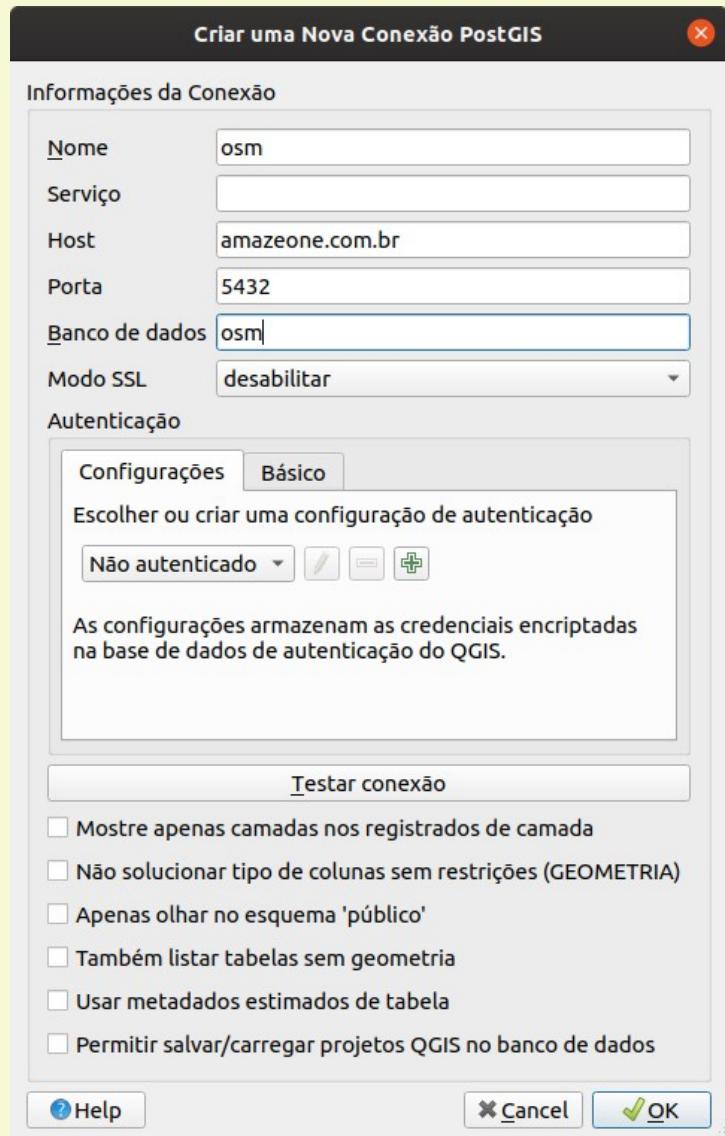
Abrindo Dados de Fontes Remotas (Banco de dados em servidores remotos)

O QGIS é uma ferramenta bastante versátil e pode abrir também dados localizados em fontes remotas do tipo banco de dados PostgreSQL-Postgis, Oracle, mySQL, DB2, etc e também dados de ArcGisMapServer, WFS, XYZ Tiles, etc.

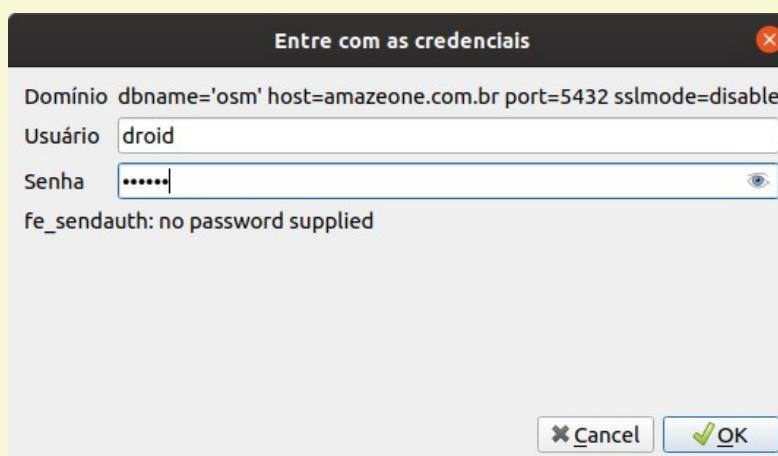
Vamos aqui mostrar como acessar um banco de dados Postgis-PostgreSQL e carregar um objeto espacial. Do lado esquerdo, no painel 'Navegador'. Clique com o botão direito do mouse e selecione **Nova Conexão...**



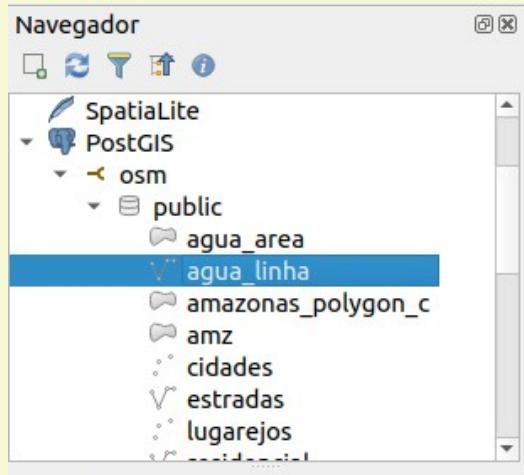
Preencha os campos conforme mostrados abaixo:



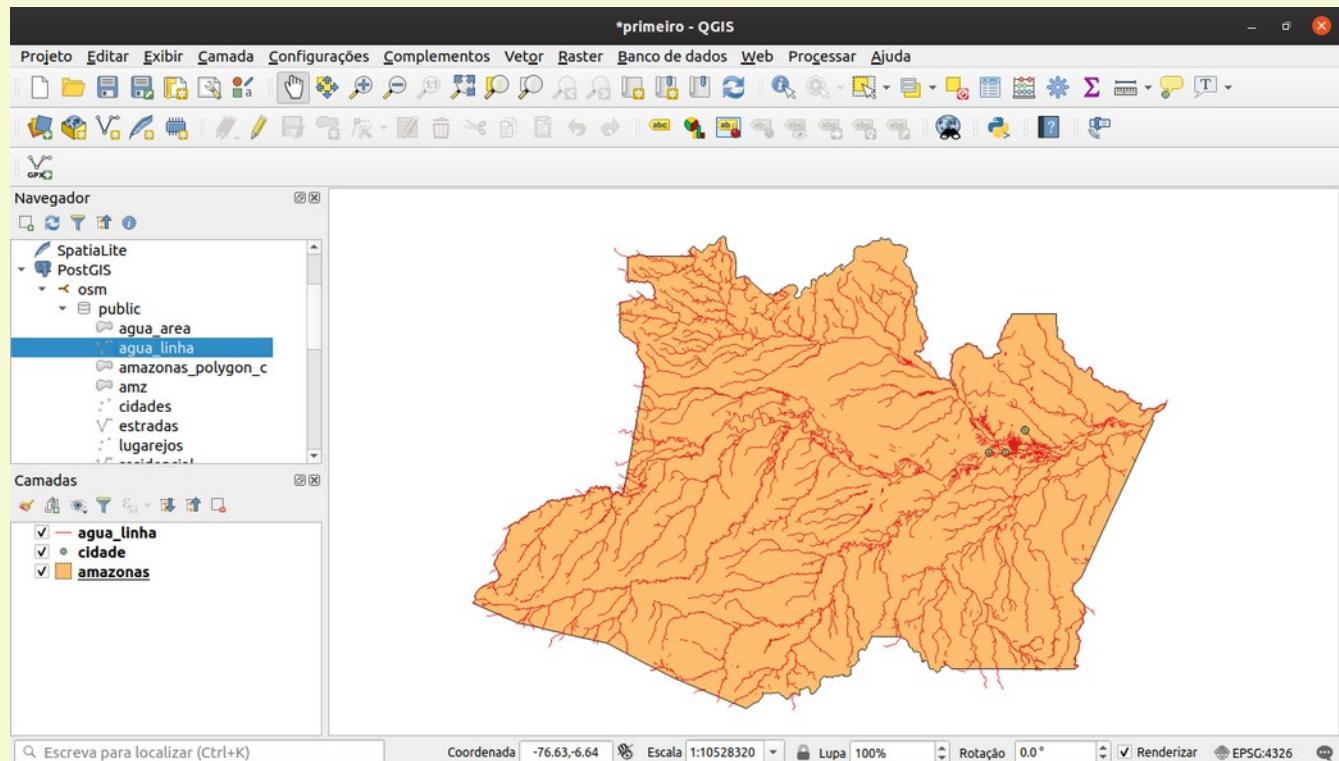
Clique na seta para baixo do Elefante do Postgis e na seta para baixo do osm, a janela abaixo vai aparecer. Entre com **droid** como usuário e **devcor** como senha, Clique **OK**.



Clique na seta para baixo em **public** e selecione, clicando duas vezes, o objeto **agua_linha** conforme abaixo para carregar o objeto:



Nosso mapa agora será algo como:



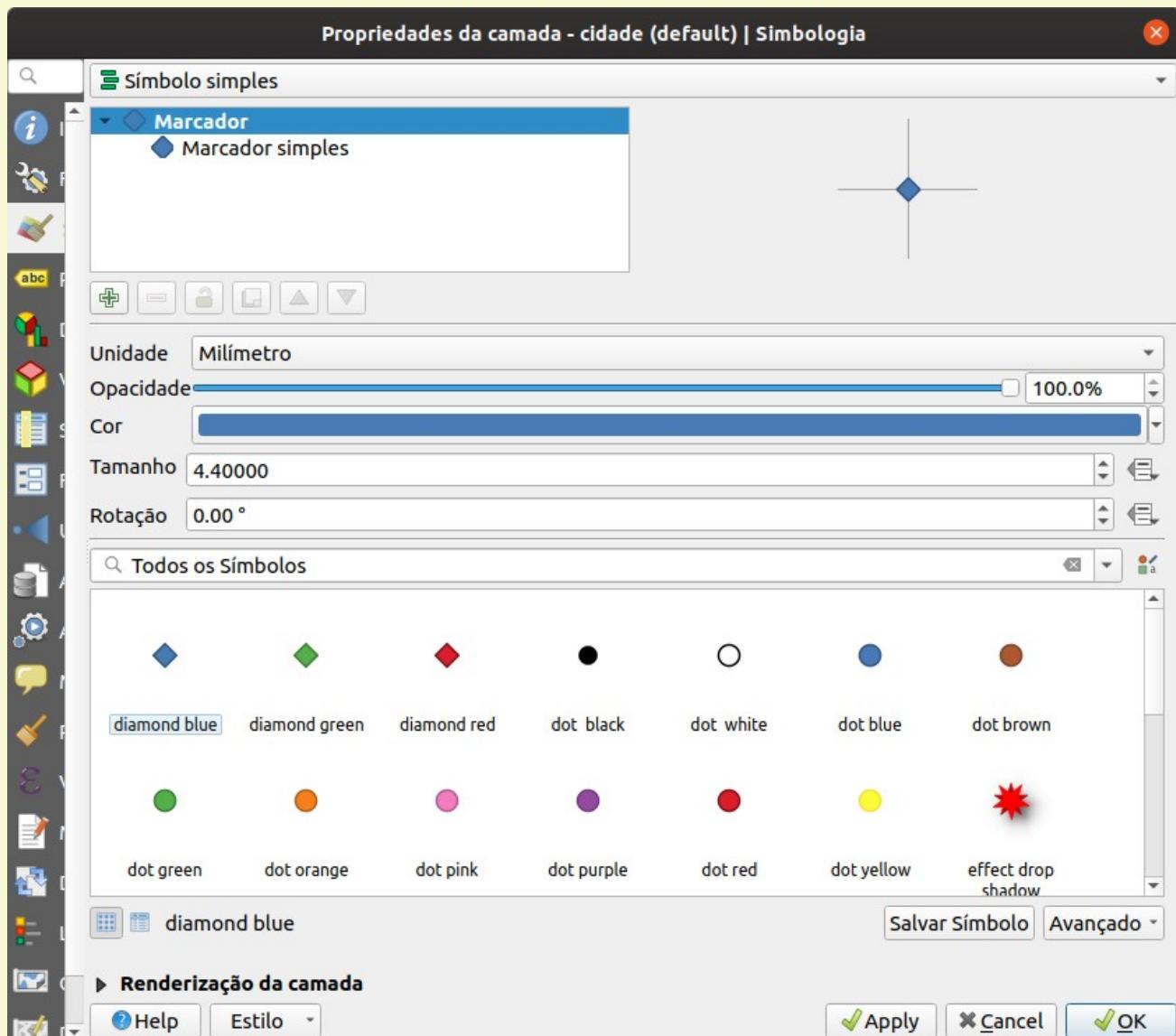
Vimos aqui como carregar objetos de dados espaciais de diversos formatos de maneira bem simples para dentro do QGIS. Salve novamente o projeto. Ao abrir novamente o projeto as credenciais do banco de dados devem ser inseridas novamente (usuário **droid** e senha **devcor**) para carregar o dado remoto.

Os tipos de objetos usados no QGIS são pontos, linhas, polígonos, multipontos, multilinhas, multipolígonos e coleções de dados (tipo misto).

Vamos agora ver como visualizar os atributos dos dados carregados e como modificar a aparência de cada um dos objetos carregados.

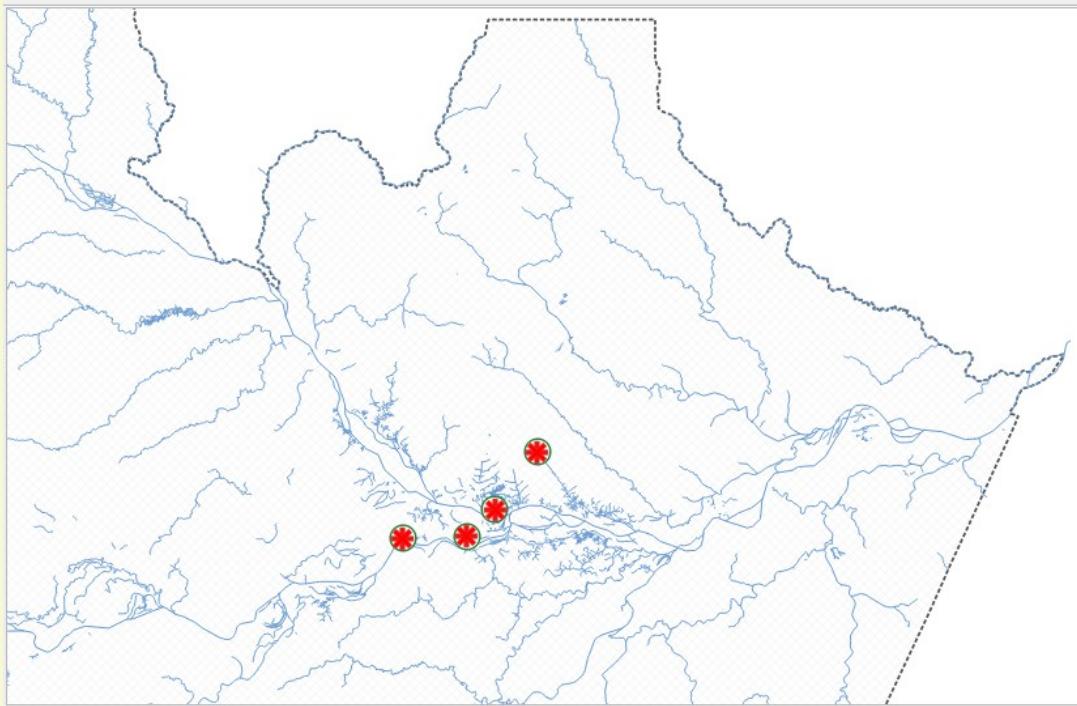
Propriedades e atributos dos objetos espaciais

Clique duas vezes na camada **cidades** no painel camadas, O painel abaixo aparecerá e nele podemos modificar a aparência da camada.



Modificamos os parâmetros de cada objeto selecionando diferentes símbolos, cores, espessuras e padrões, podemos também personalizar símbolos usando o botão **Salvar Símbolo**.

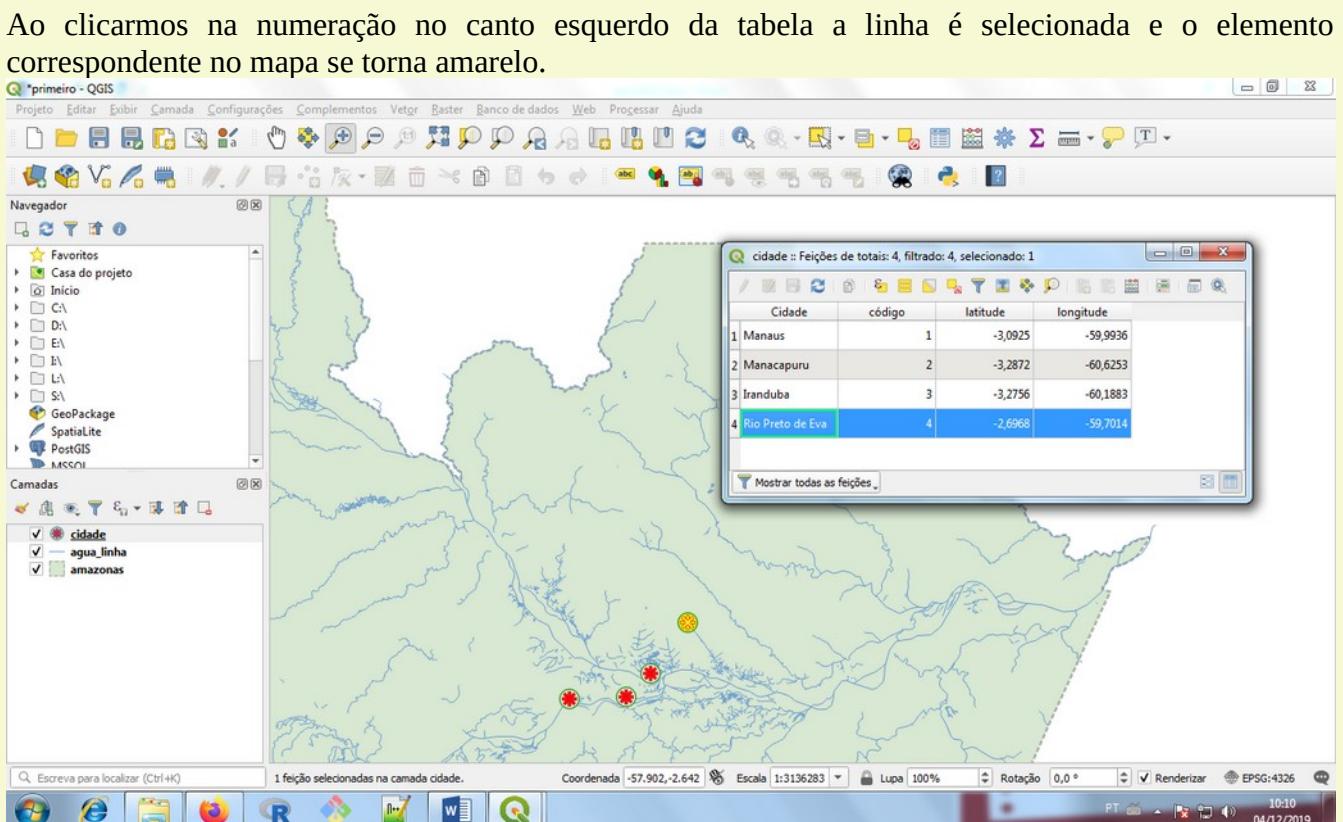
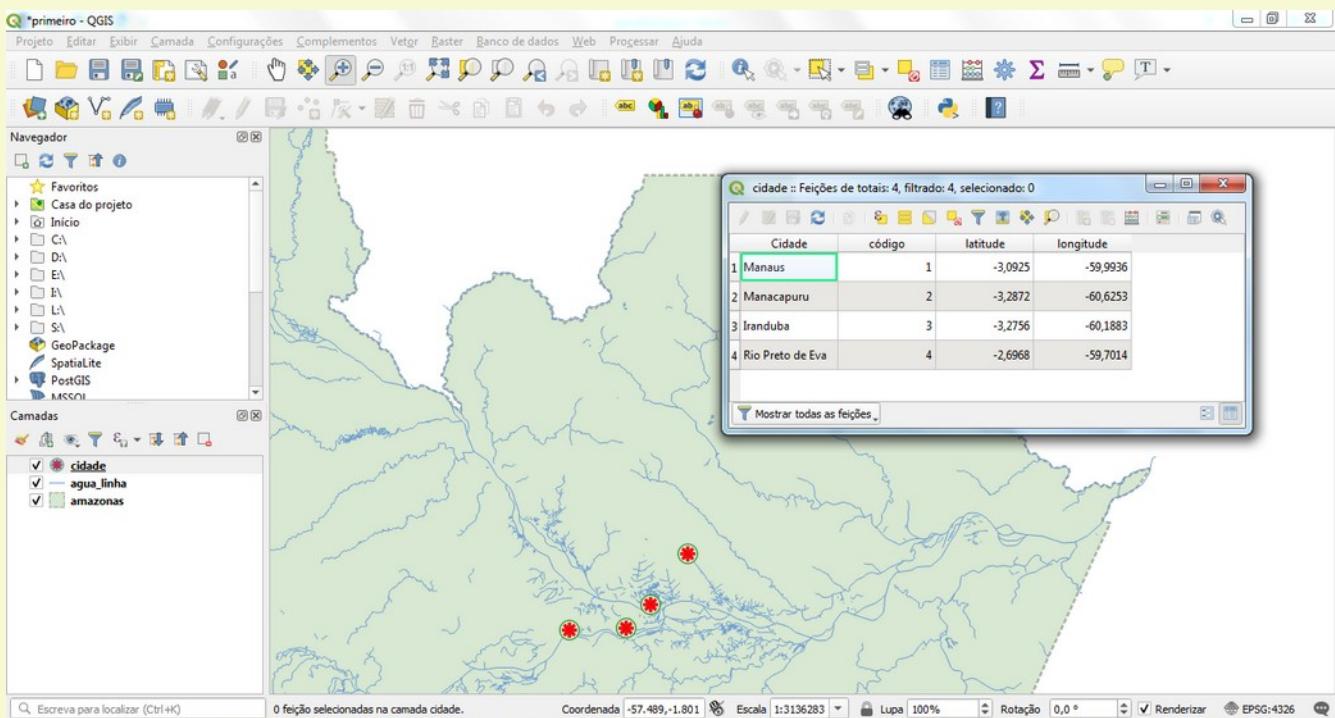
Vamos efetuar um exercício e transformar a aparência de cada objeto até obtermos um resultado semelhante ao da imagem abaixo:



Abaixo mostramos como ver a tabela de atributos das camadas abertas. Obra o projeto **primeiro.qgz** caso não esteja aberto e no painel de camadas selecione a camada amazonas, Vá no menu **Camada > Abrir tabela de Atributos** ou pressione F6. A seguinte janela irá aparecer.

MSLINK	MAPID	CODIGO	AREA_1	PERIMETRO_	GEOCODIGO	NOME	AREA_TOT_G	Perimeter	Area	Hectares
13	99	13	1570745.679999...	8133.839710000...	13	AMAZONAS	1570745.679999...	8117.091709620...	1570808.983209...	157080898.3210...

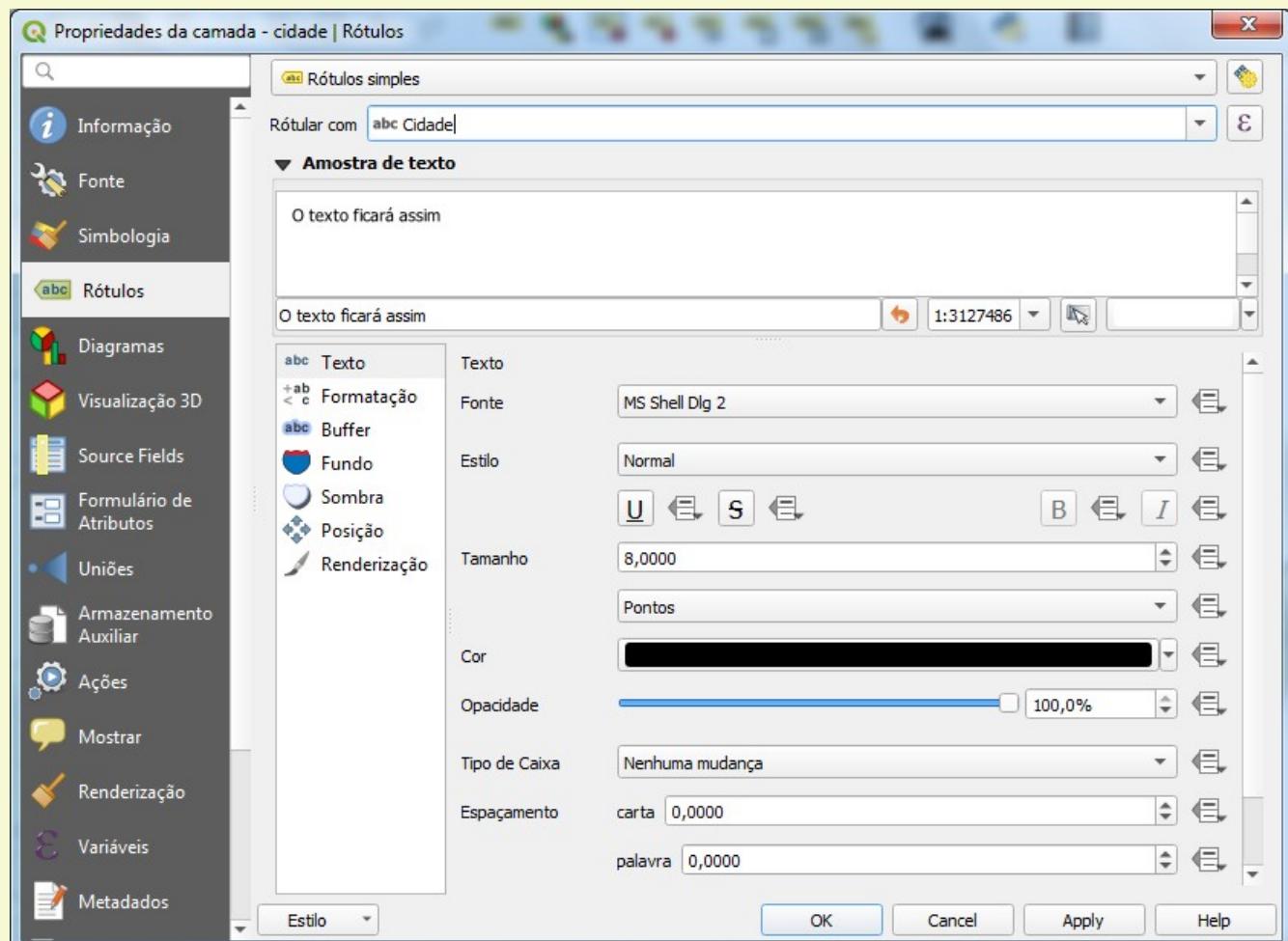
A tabela mostra os atributos da camada amazonas, nesse caso constituída de um único elemento. Feche a janela da tabela e vamos repetir o processo usando a camada cidades. Selecione cidades e pressione F6, o seguinte de aparecer Ajuste as janelas conforme abaixo para mostrar como funciona o processo de selecionar elementos:



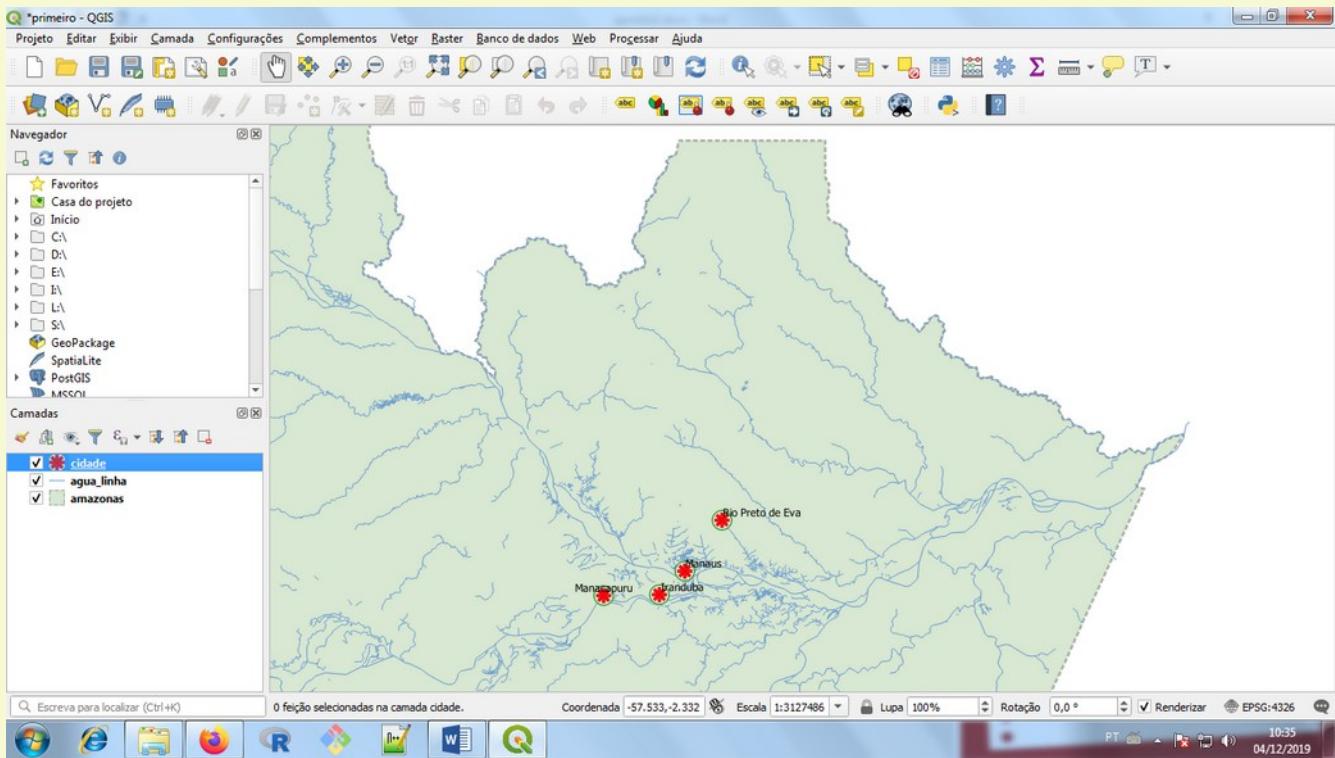
Abaixo vemos as opções de seleção da barra de ferramentas da janela da tabela de dados.

-  Seleciona tudo
-  Inverte a seleção
-  Limpa toda a seleção
-  Seleciona elementos usando expressões

Vamos agora mostrar como adicionar rótulos nos elementos de uma camada. Selecione e clique duas vezes na camada cidades. No painel lateral esquerdo selecione Rótulos e selecione Rótulo Simples na caixa de seleção, preencha conforme abaixo e clique em Ok:



Cada cidade da camada apresentará um rótulo com o nome da cidade.



1.3 Carregando dados Raster

QGIS trabalha com dados no formato raster (imagem) também. Dados do tipo raster são uma representação de quantidades ou grandezas que variam ou não em intervalos regulares (grid), em vez de uma grandeza única pontual ou ao longo de uma linha ou numa área específica como é um dado vetorial.

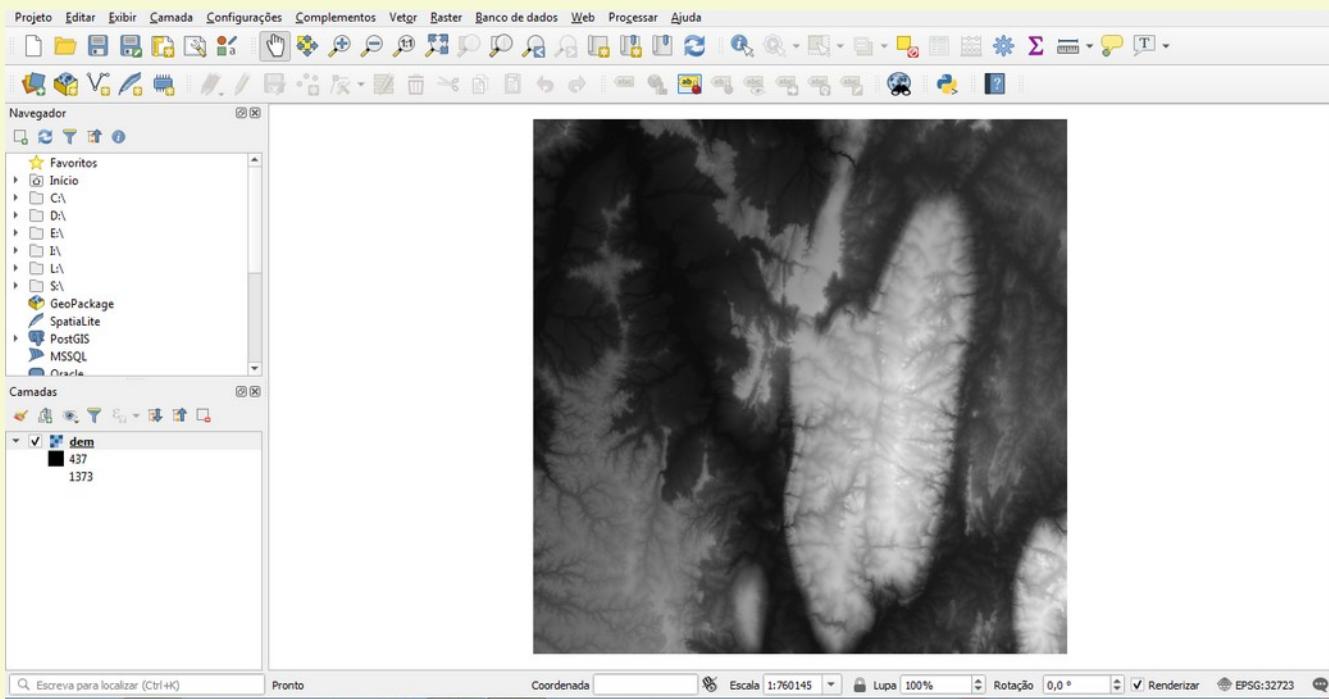
Dados do tipo raster podem apresentar valores de uma única grandeza no seu grid ou várias grandezas (separadas bandas). Um exemplo de bandas de valores são os dados multiespectrais de sensoriamento remoto. Um exemplo de dados de simples grandeza são os modelos digitais de elevação.

Dados pontuais do tipo vetorial podem ser convertidos em dados do tipo raster usando interpolações matemáticas ou estatísticas que calcula ou estima a distribuição dos valores em intervalos regulares de acordo com o grid a ser criado.

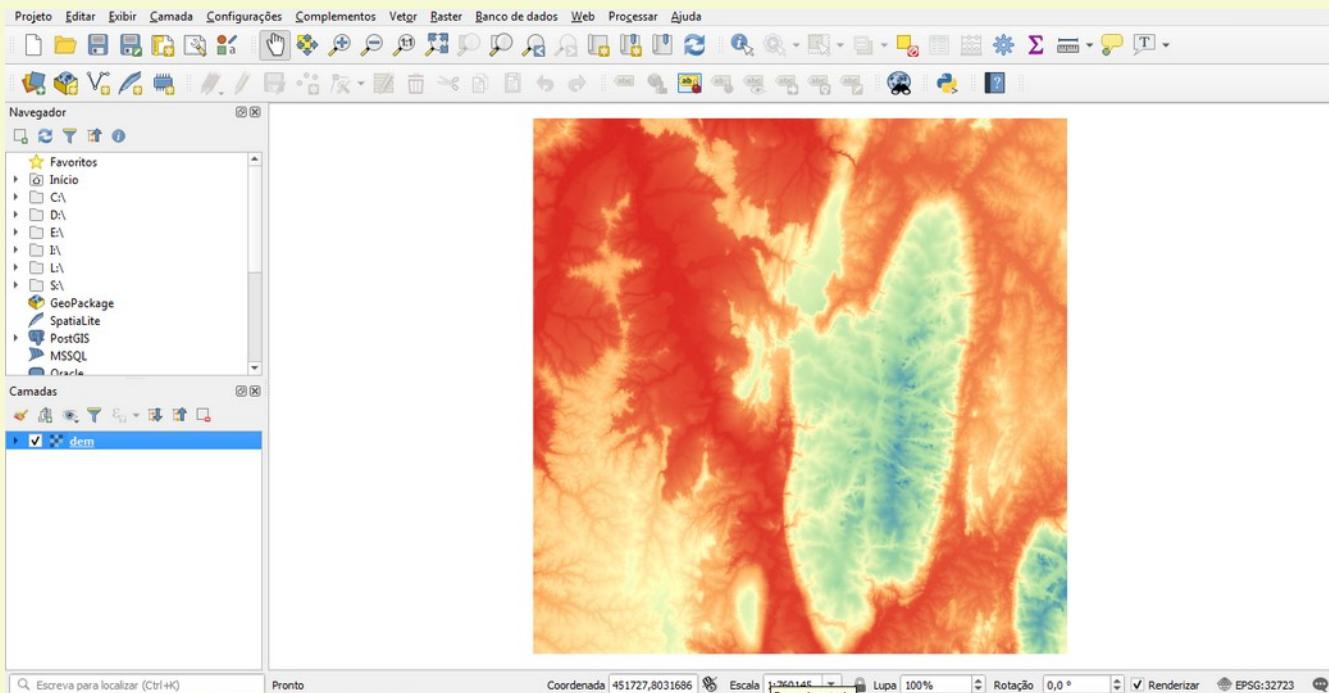
Vamos aqui ver como carregar uma imagem de um modelo digital de elevação e gerar um sombreamento para realce de relevo. Depois vamos carregar uma imagem Sentinel2 com 3 bandas no espectro do visível da mesma área.

Abra o QGIS e carregue a imagem raster dem.tif **Camada > Adicionar Camada > Raster** ou **Ctrl+Shift +r**.

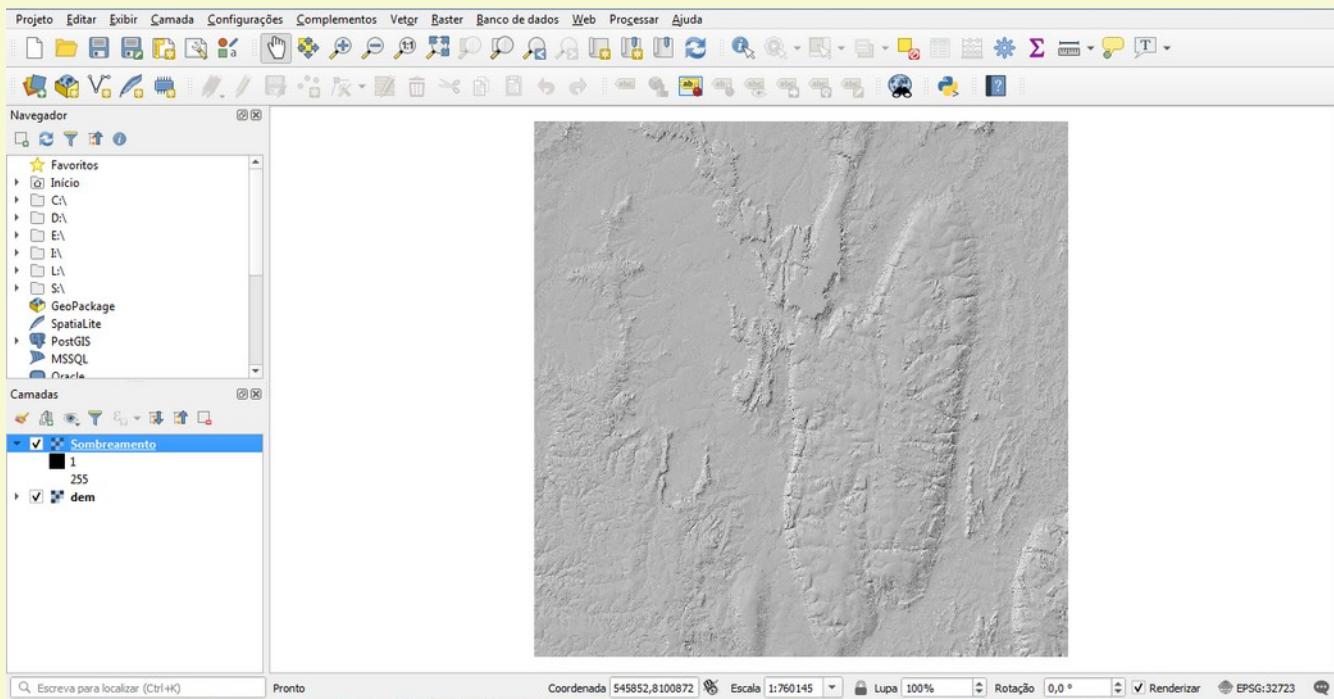
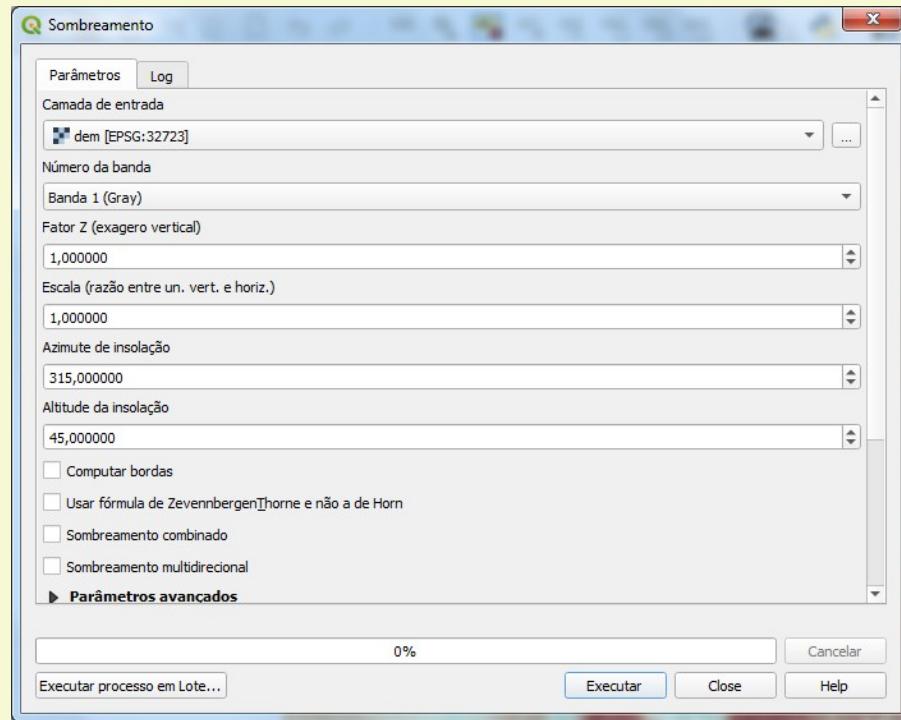
E proceda acionando ... para navegar ao local do arquivo **dem.tif**. Em seguida clique em **Adicionar** e **Close**. A objeto raster deverá ser carregado conforme mostrado abaixo.



No painel Camadas clique duas vezes em dem e vamos modificar as cores de cinza para uma rampa de cor já predefinida. Na janela que se abriu selecione **Simbologia** no painel da esquerda e selecione **Paletizado/Valores Únicos** no campo **Tipo de renderização**. Selecione **Spectral** no campo Gradiente de Cores e pressione o botão **Classifica**. Após isso clique em **OK**. Nosso DEM aparecerá conforme a imagem abaixo.

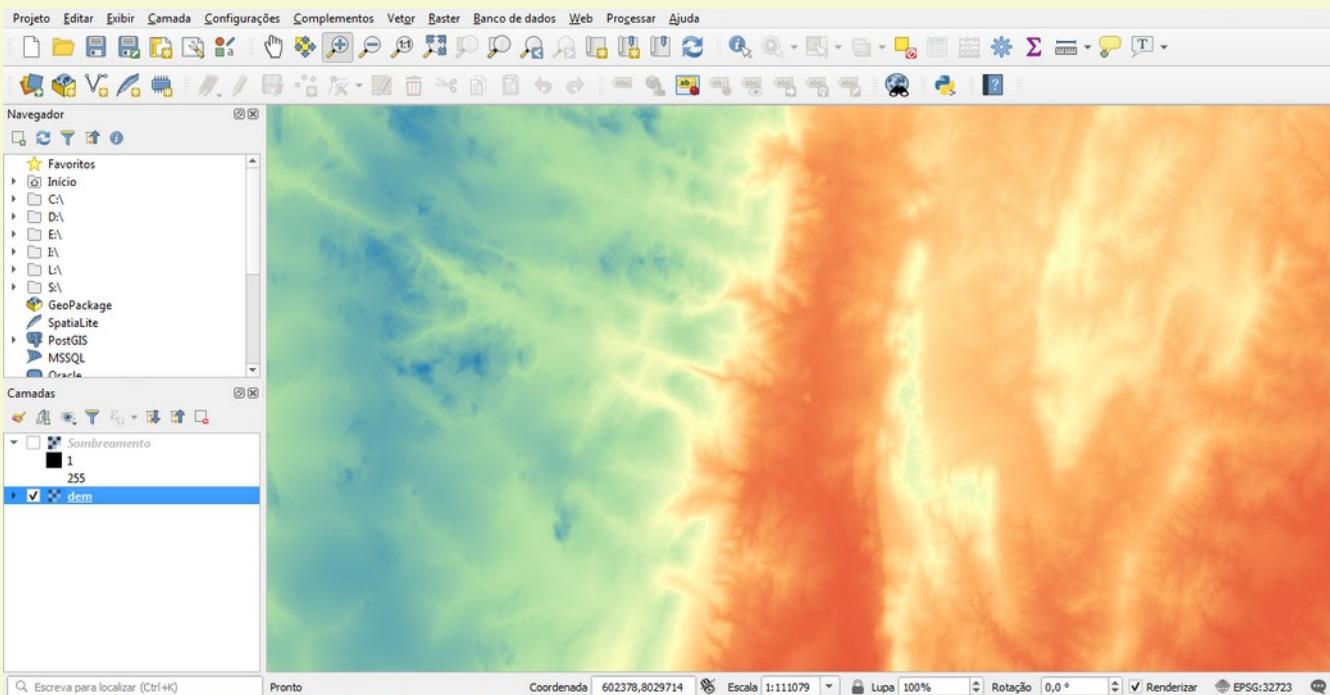


Vamos agora processar o DEM para gerar um sombreamento para realçar o relevo da imagem. No menu selecione **Raster > Análise > Sombreamento** e a janela aparecerá. Clique em **Executar** com os parâmetros apresentados e a imagem Sombreamento será criada.

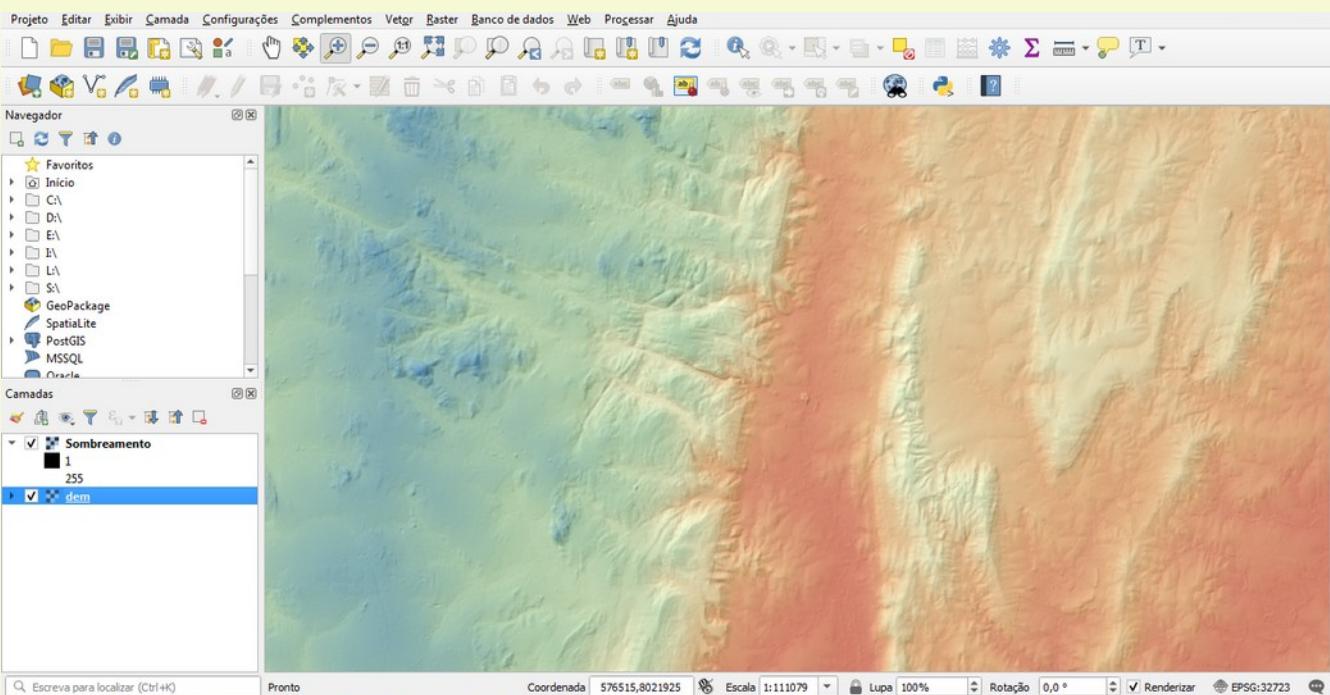


Clique duas vezes em sombreamento e selecione **Transparência** na esquerda, entre com o valor 40% no primeiro campo e clique em **OK**. De um zoom numa região com diferença de relevo e compare o resultado.

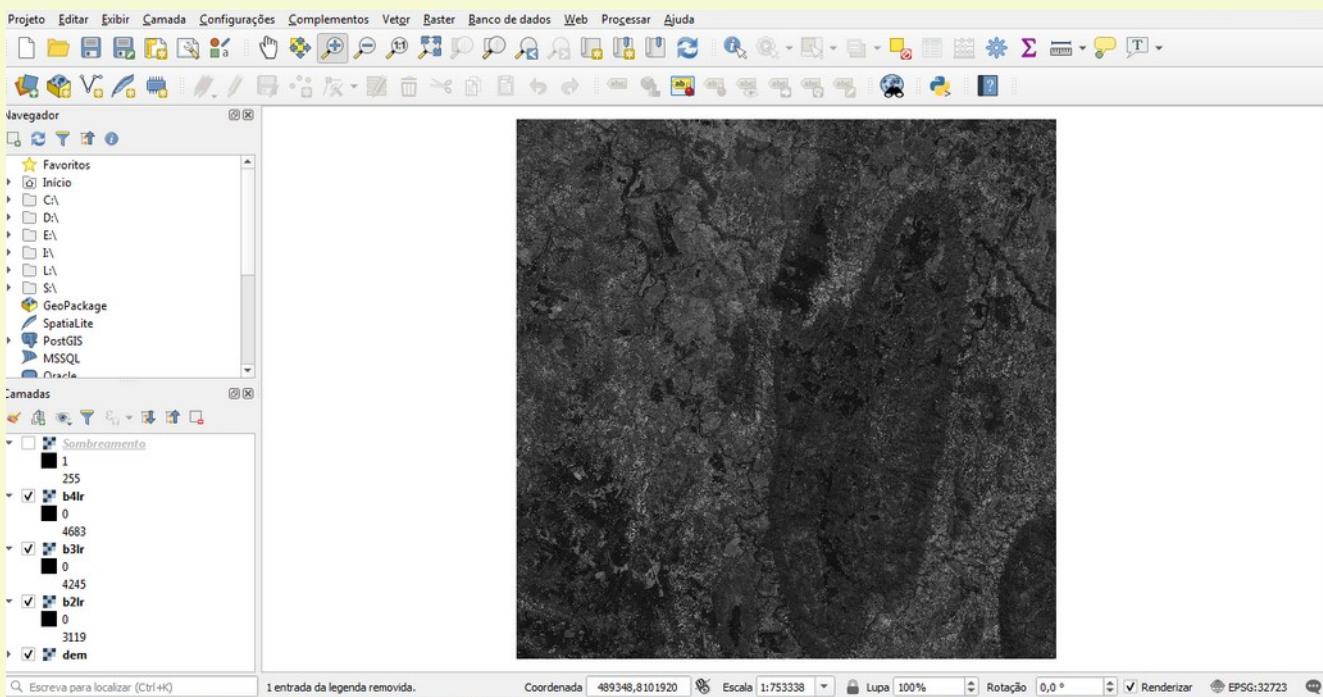
Sem sombreamento:



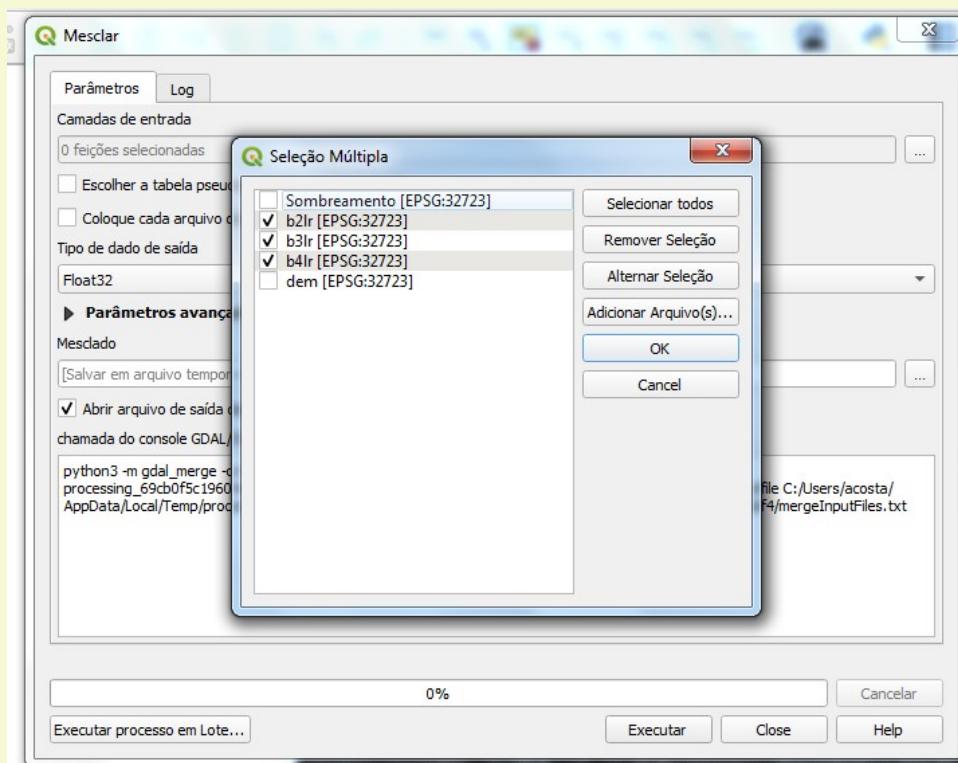
Com Sombreamento:



Vamos agora carregar as bandas 2, 3 e 4 do Sentinel2 desta área. Carregue as três bandas (Crl+Shift+r) e adicione elas ao projeto.

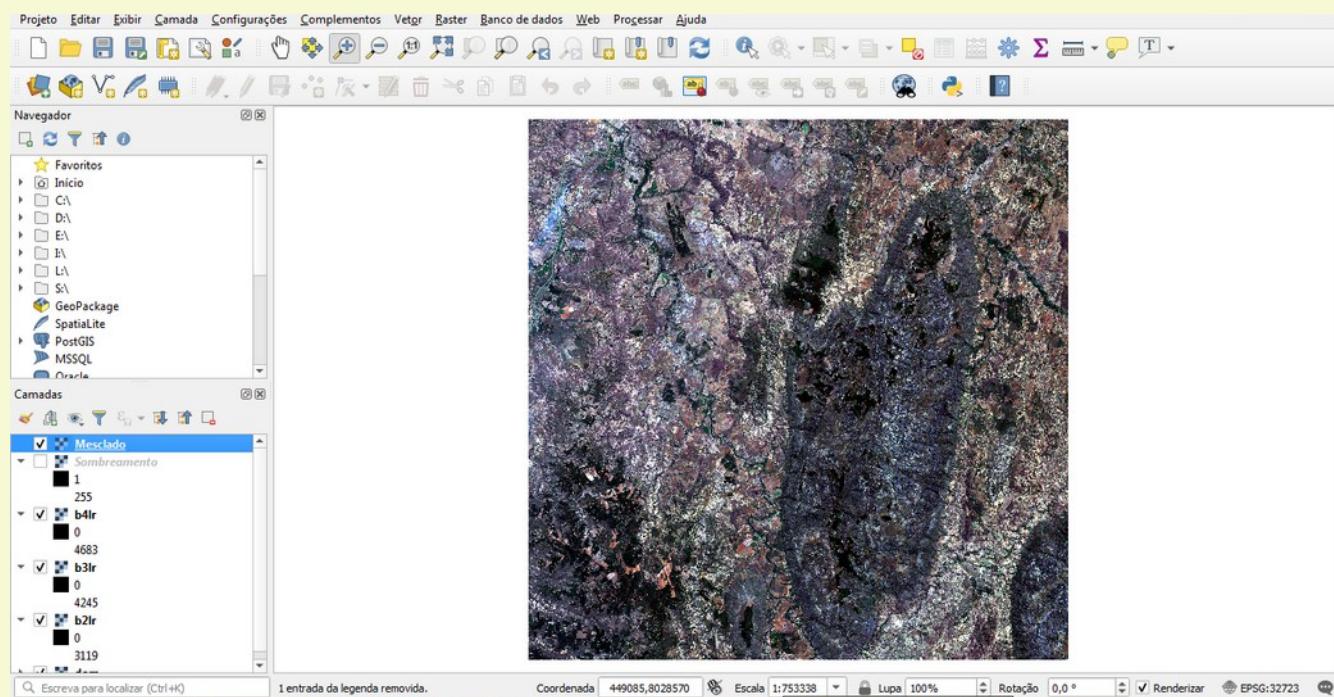


Combinamos estas três bandas para criar uma composição de cor verdadeira (RGB) usando **Raster > Miscelânea > Mesclar**. Selecione as três bandas recém carregadas clicando no ... e marcando elas conforme abaixo.

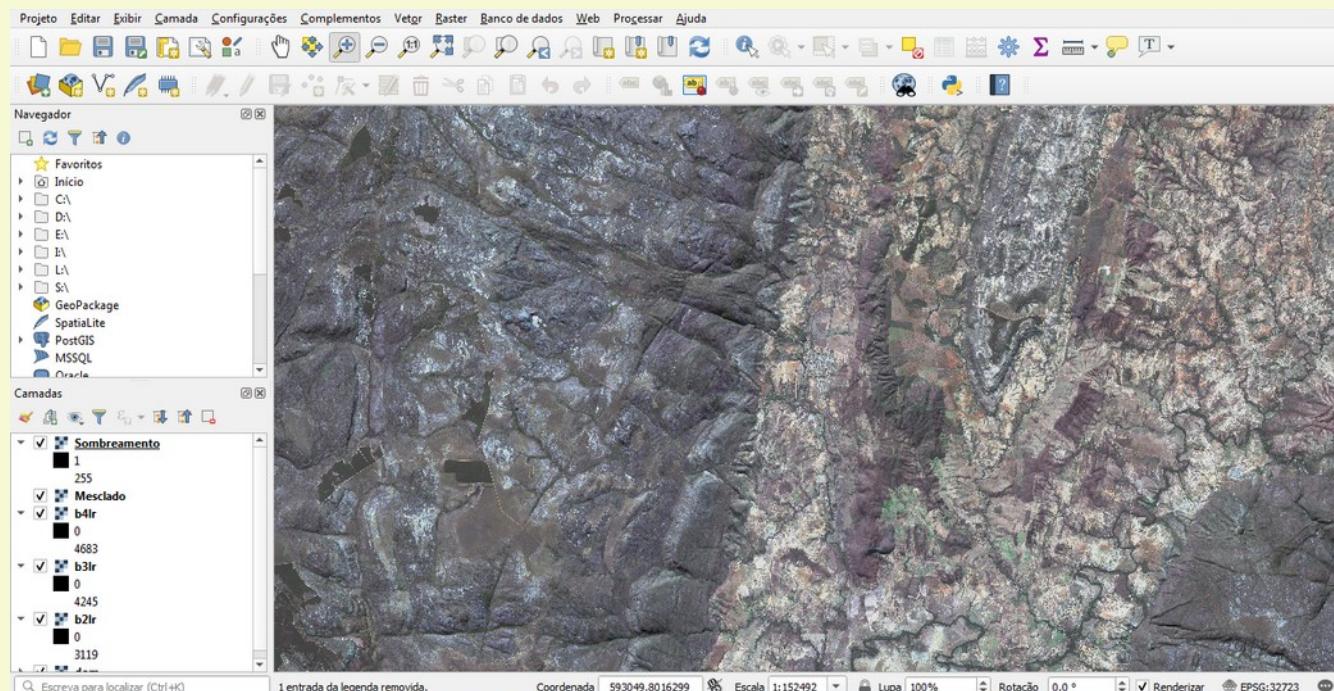


Clique **OK** na janela de Seleção Múltipla e em seguida coloque cada arquivo de entrada em uma banda separada. Clique **Executar** e **Close** e uma nova imagem raster chamada Mesclado.

Vamos clicar duas vezes em Mesclado e colocar banda 3 no canal vermelho e Banda 1 no canal Azul e clicar em OK. A nossa imagem deverá aparecer assim.



Vamos mover Mesclado para baixo do Sombreamento no painel Camadas, ativar Sombreamento e dar um zoom numa área. Veremos um resultado assim:

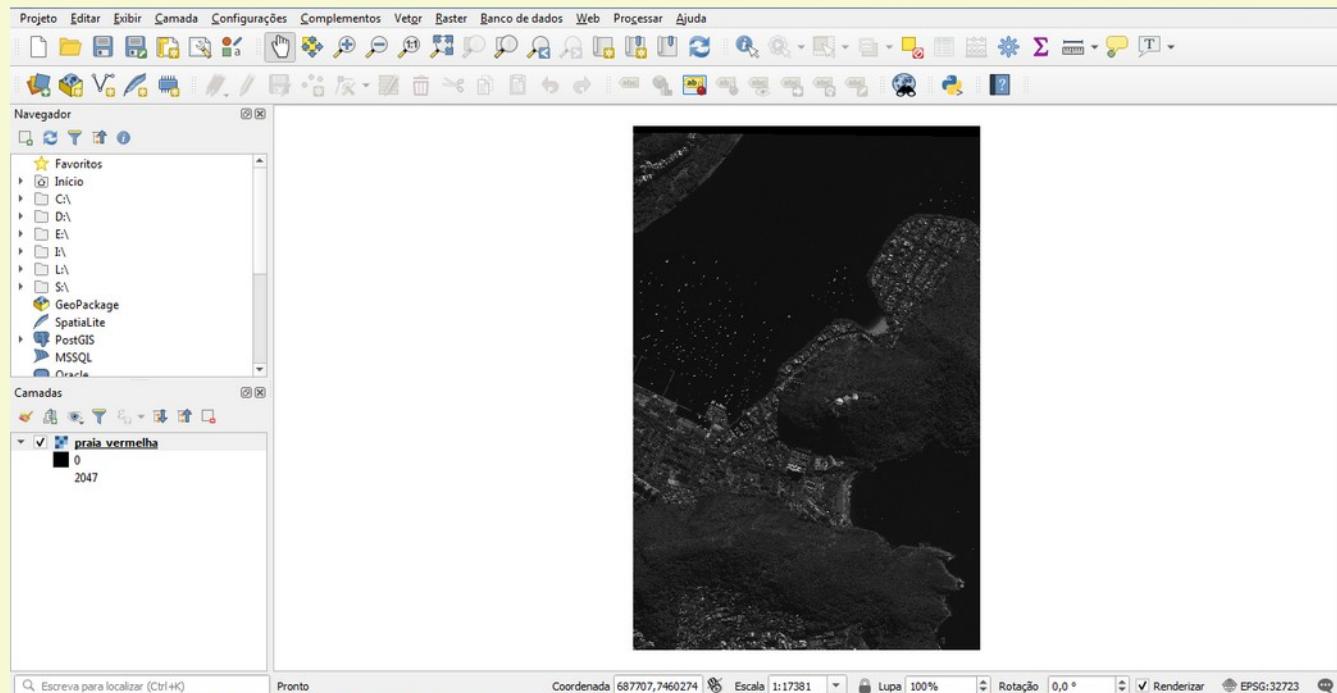


Grave este projeto como **imagens.qgz** para usarmos depois. Muito mais pode ser feito com imagens Raster, essa foi somente uma introdução de como manipular imagens raster. Vamos agora ver com criar vetores e rasters em QGIS.

1.4 Criando Dados Vetoriais

Podemos criar novas camadas do tipo ponto, linha e polígono usando o QGIS. Vamos fazer isso usando uma imagem de alta resolução georreferenciada como base e criar camadas para elementos dessa imagem.

Obra o QGIS e carregue a imagem praia_vermelha.TIF.



Essa imagem da Praia Vermelha e Morro da Urca servirá de base para criarmos camadas.

Entre em **Camada > Criar Nova Camada > Shapefile** para criar uma nova camada.

Vamos inicialmente criar uma camada do tipo ponto.

Primeiro criamos o arquivo selecionando o botão ..., navegue até a pasta desejada e crie um arquivo com o nome **poi.shp**

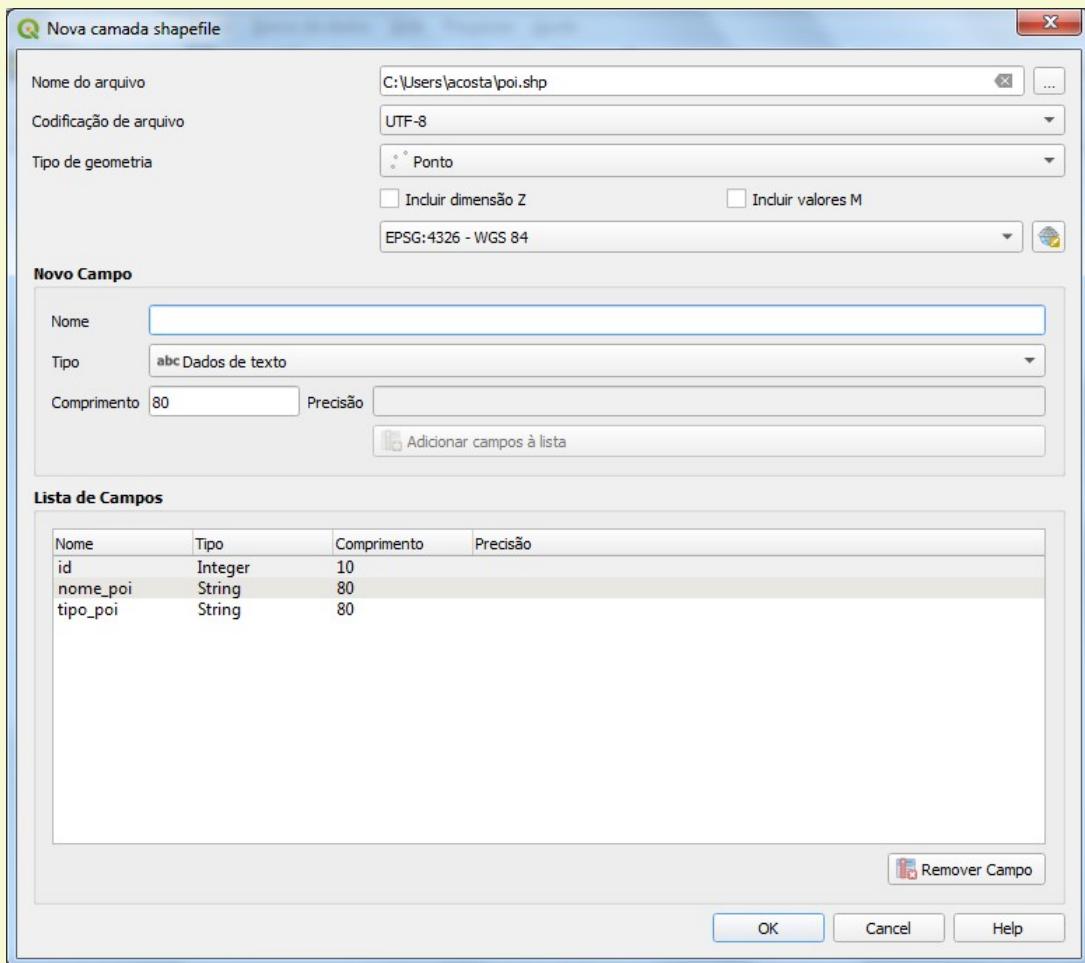
No campo **Tipo de geometria** deixe como **Ponto**.

Apesar da projeção de imagem raster ser EPSG 32723 vamos deixar a projeção dos nossos pontos como EPSG 4326, QGIS faz a transformação automaticamente.

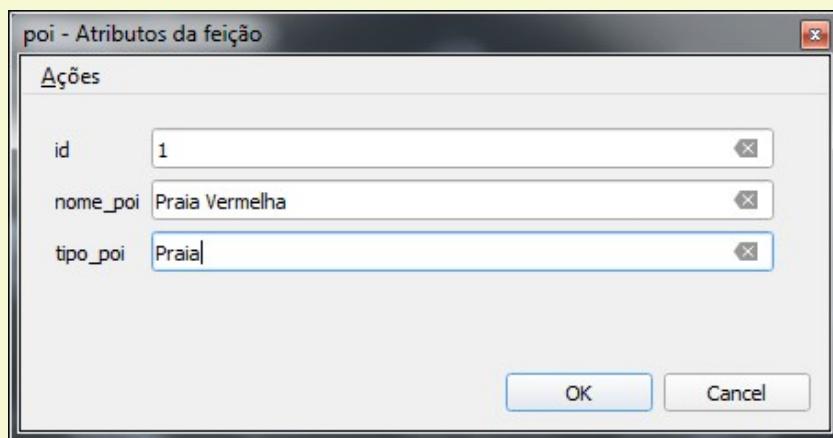
Agora vamos criar a nossa tabela de atributos da camada. O atributo ID já é criado automaticamente.

Entre no campo **Nome** o valor **nome_poi**, em **Tipo** selecione Dados de Texto e clique em **Adicionar campo a Lista**. O atributo será adicionado. Repita o processo e adicione um atributo de nome **tipo_poi** como dado de texto.

O painel deverá estar como abaixo, Clique em **OK**:



Uma nova camada com o nome poi aparecerá no painel de Camadas. Estamos prontos para adicionar novos pontos nela. O processo é feito pelo menu Camada > Alternar edição, estamos prontos para adicionar pontos. Fazemos isso pelo menu **Editar > Adicionar Ponto** ou pelo barra de ferramenta clicando em . O cursor se modifica e você já pode clicar na imagem na posição do primeiro ponto. Vamos clicar na Praia Vermelha. Ao clicar uma janela se abre, Preencha os atributos com ID =1, nome_poi= Praia Vermelha, e tipo_poi = Praia. Deverá ficar como abaixo, Clique OK para finalizar.



Vamos repetir o processo adicionando 4 novos pontos. Um ponto em cada estação do bondinho na imagem e um ponto na marina do Iate Clube.

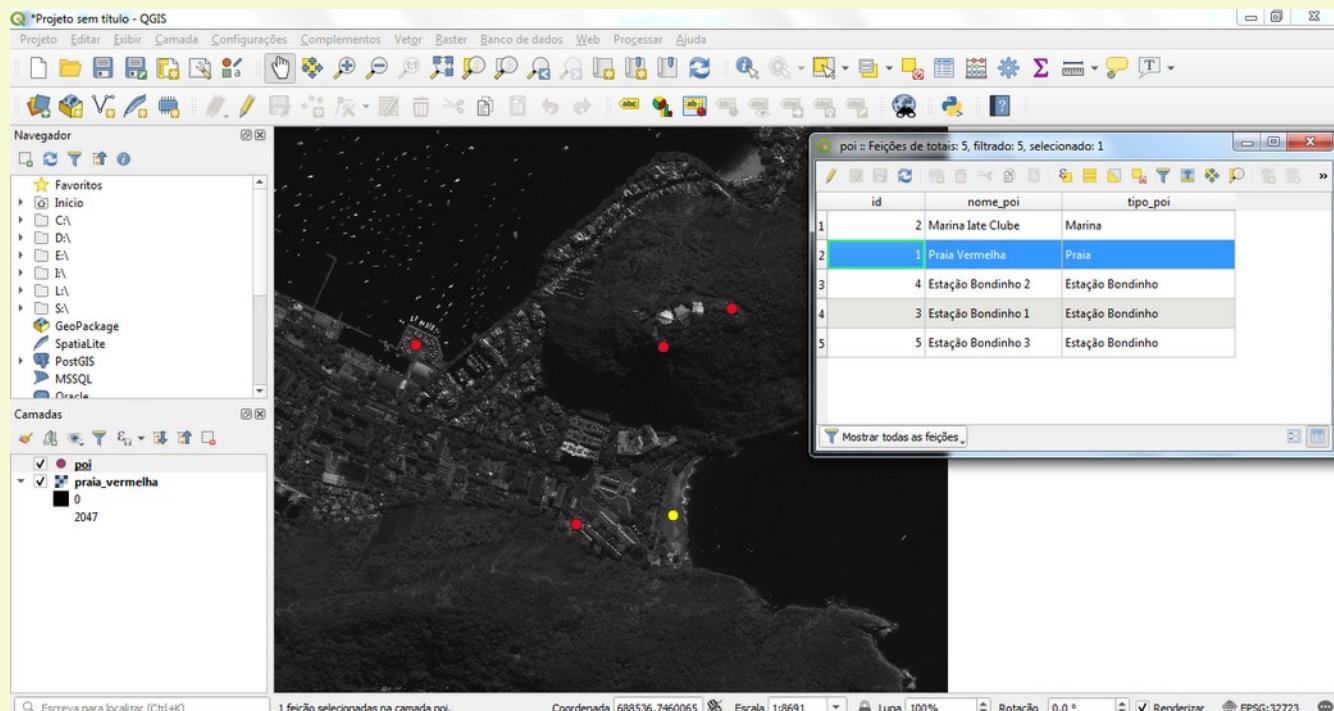
Atributos com ID =2, nome_poi= Mariana Iate Clube e tipo_poi = Marina

Atributos com ID =3, nome_poi= Estação Bondinho 1 e tipo_poi = Estação Bondinho

Atributos com ID =4, nome_poi= Estação Bondinho 2 e tipo_poi = Estação Bondinho

Atributos com ID =5, nome_poi= Estação Bondinho 3 e tipo_poi = Estação Bondinho

Ao terminar de adicionar os pontos clique em **Camada > Alternar edição** para finalizar e clique em **Save** para salvar a adição dos pontos. Abra agora a tabela de dados e selecione um dos pontos. Teremos algo como o mostrado abaixo.

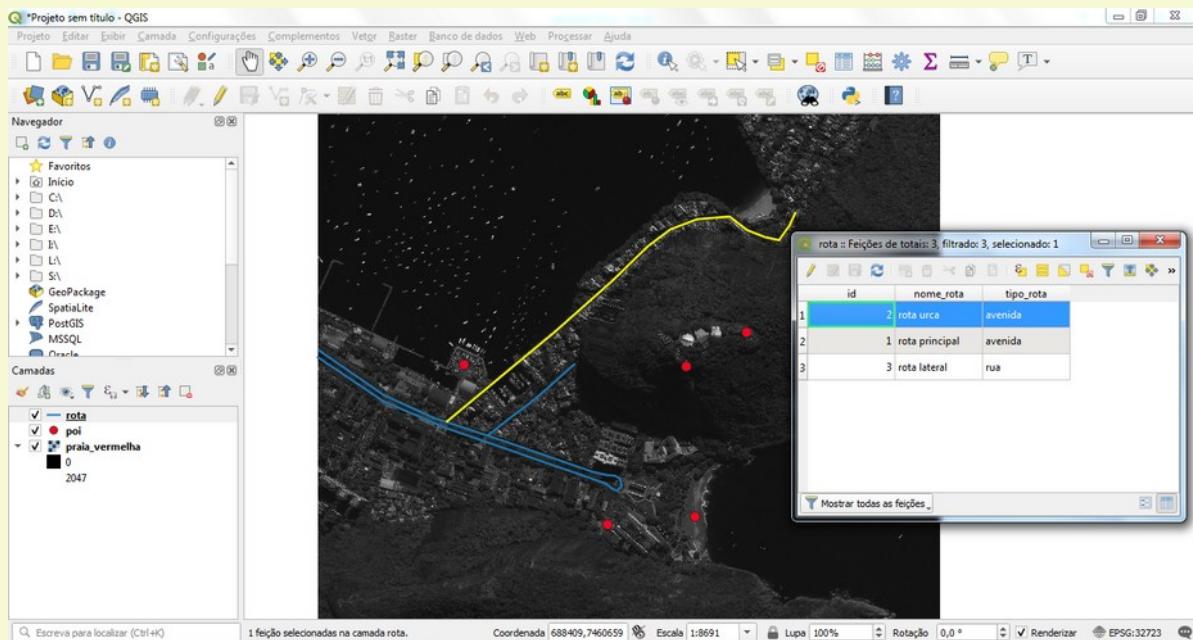


Acabamos de criar uma camada do tipo pontos. Vamos agora repetir o processo criando uma camada do tipo linha e outra do tipo polígono.

Entre em **Camada > Criar Nova Camada > Shapefile** para criar uma nova camada e crie uma camada do tipo linha com o nome de rota com os atributos nome_rota e tipo_rota.

Clique em e digite as linhas clicando com o botão esquerdo do mouse, ao terminar uma avenida clique no botão direito do mouse e entre com os atributos desta linha, repita novamente para digitar duas novas linhas.

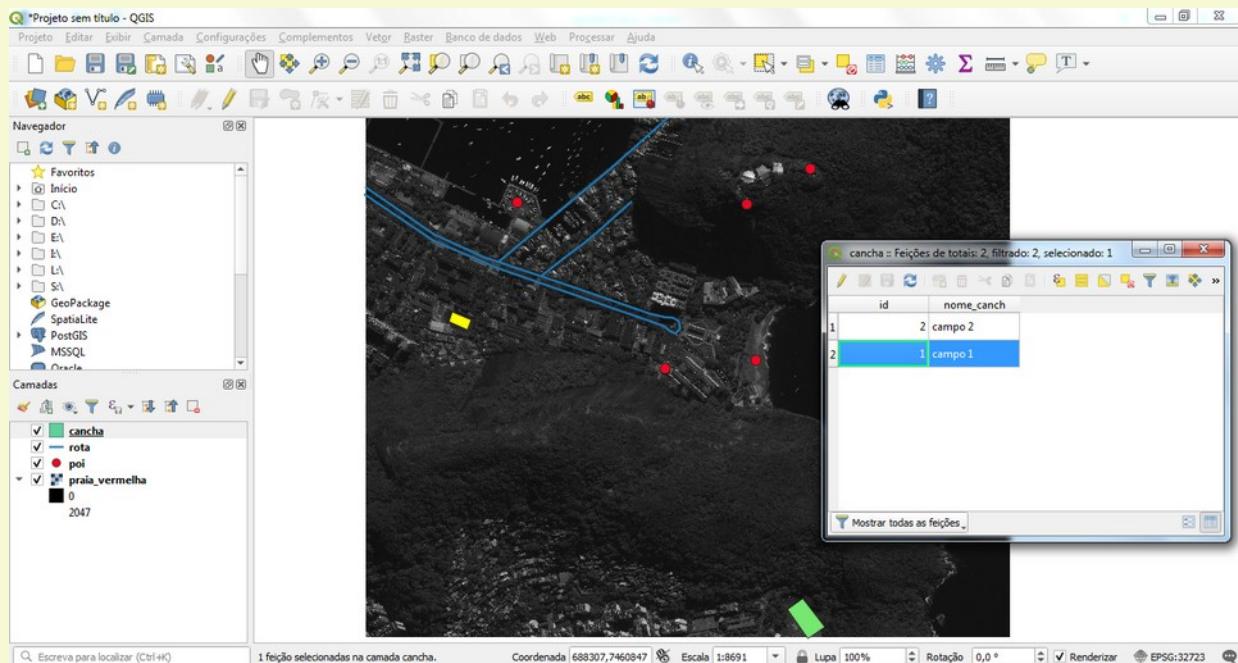
Ao final , clique novamente em **Altenar Edição** e grave o recém criado clicando em **Save**. Abra a janela de tabela de atributos da camada e algo similar ao mostrado abaixo deverá aparecer.



Finalizaremos agora criando uma camada do tipo polígono. Entre em **Camada > Criar Nova Camada > Shapefile** para criar uma nova camada e crie uma camada do tipo polígono com o nome de cancha com o atributo nome_canch.

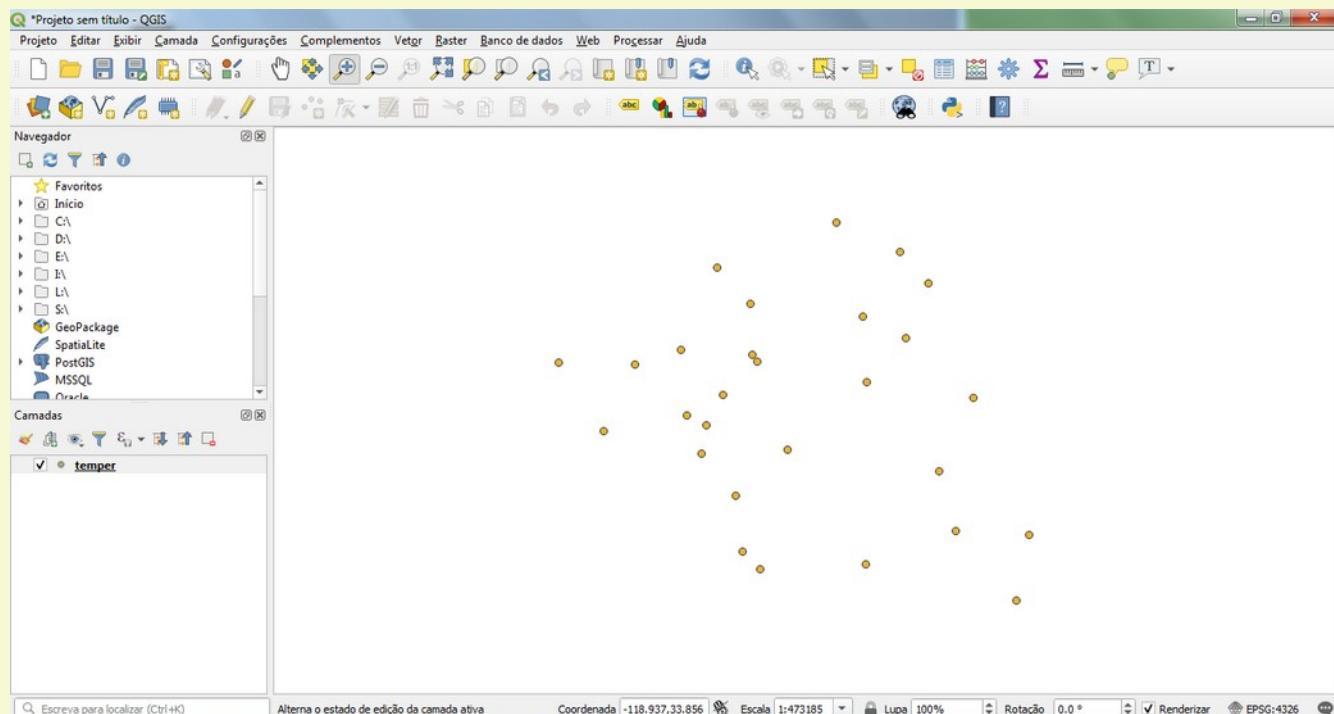
Clique em e digite as linhas clicando com o botão esquerdo do mouse, ao terminar um dos campos de futebol clique no botão direito do mouse e entre com os atributos deste polígono, repita novamente para digitar o outro polígono.

Ao final , clique novamente em **Altenar Edição** e grave o recém criado clicando em **Save**. Abra a janela de tabela de atributos da camada e algo similar ao mostrado abaixo deverá aparecer.

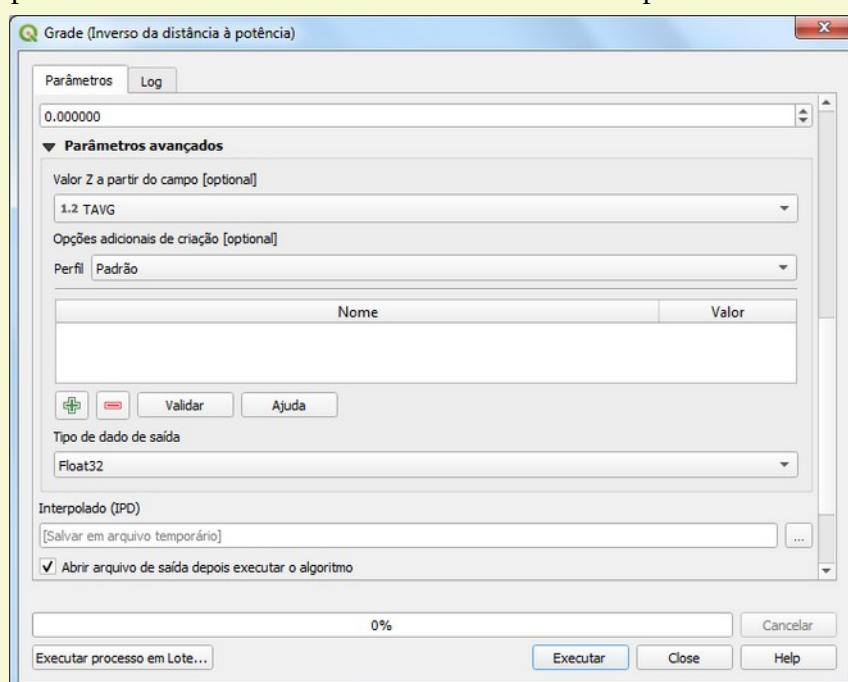


1.5 Criando Raster a partir de Pontos Vetoriais

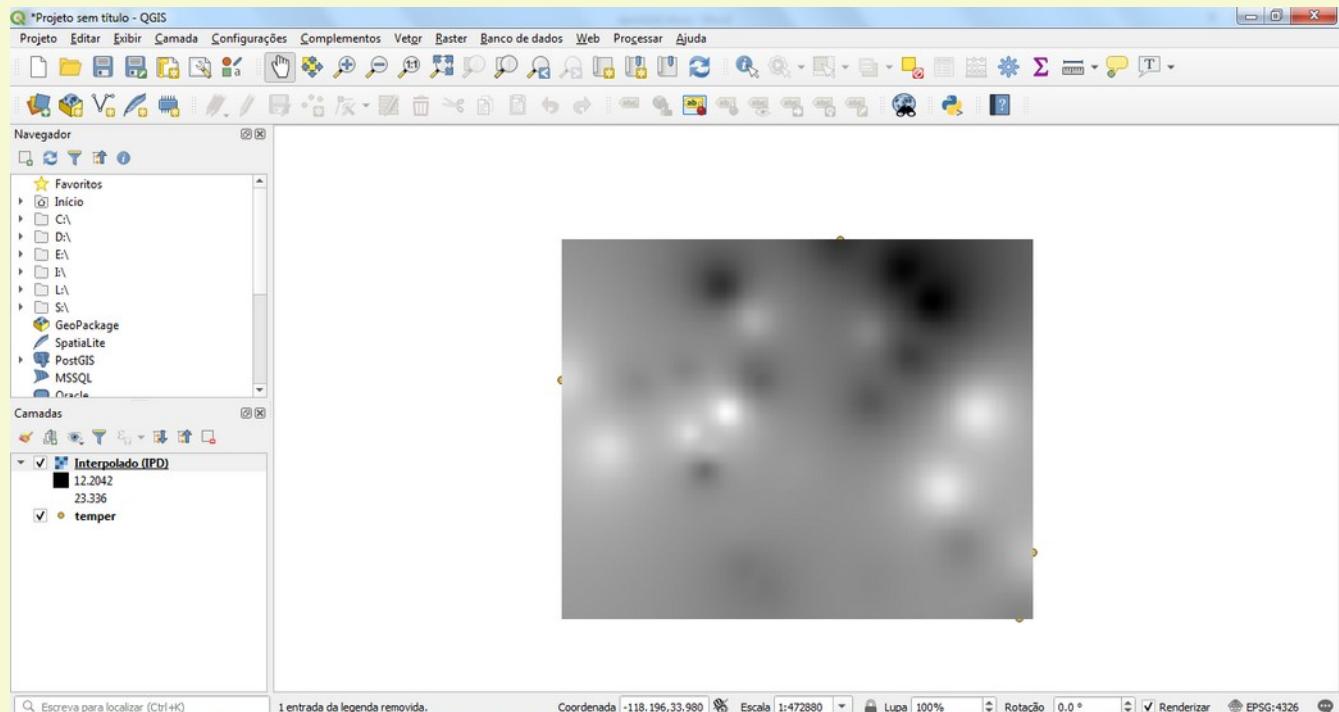
Podemos criar imagens raster a partir de dados pontuais usando interpolação dos dados. Vamos mostrar aqui os passos de como criar este raster. Primeiramente vamos abrir o arquivo **temper.shp** com dados pontuais de temperatura no dia 13 de Novembro de 2015 na região de Los Angeles.



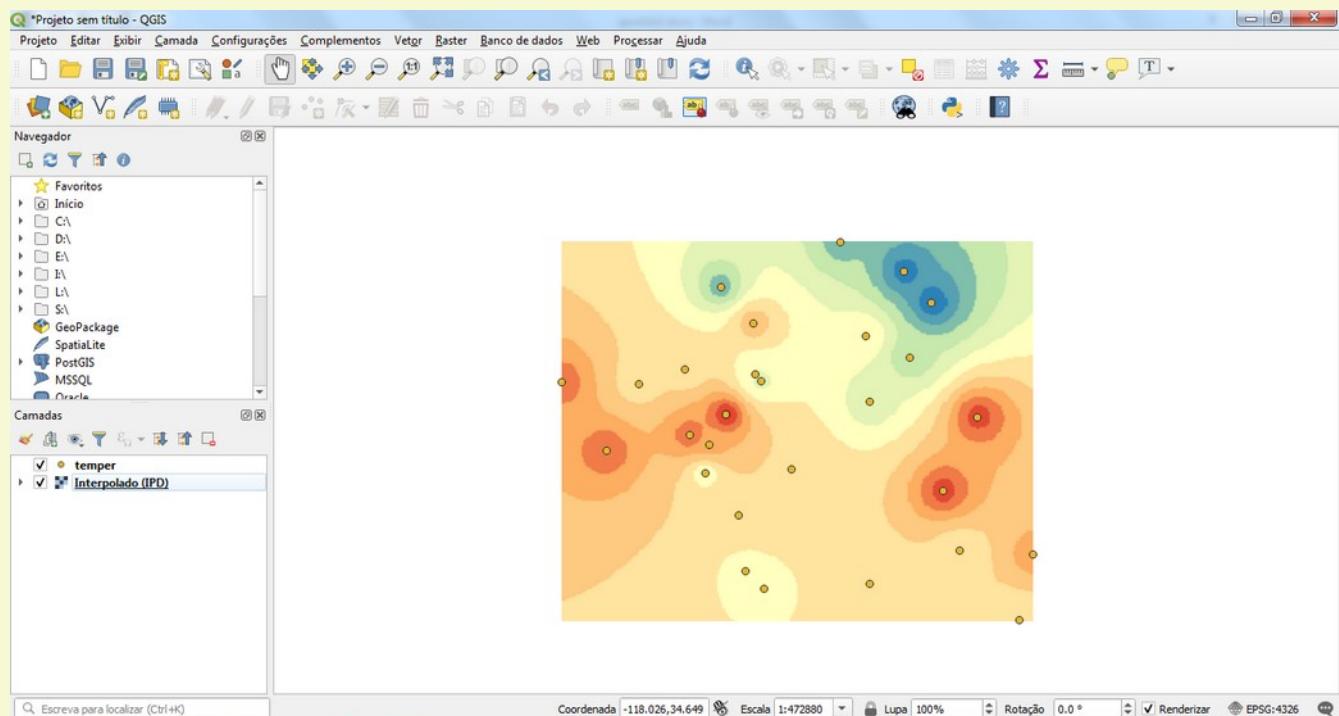
Selecione o menu **Raster > Análise > Grade** (Inverso da distância a Potência). Vamos manter os valores padrões apresentados selecionando somente TAVG no campo Valor Z conforme abaixo.



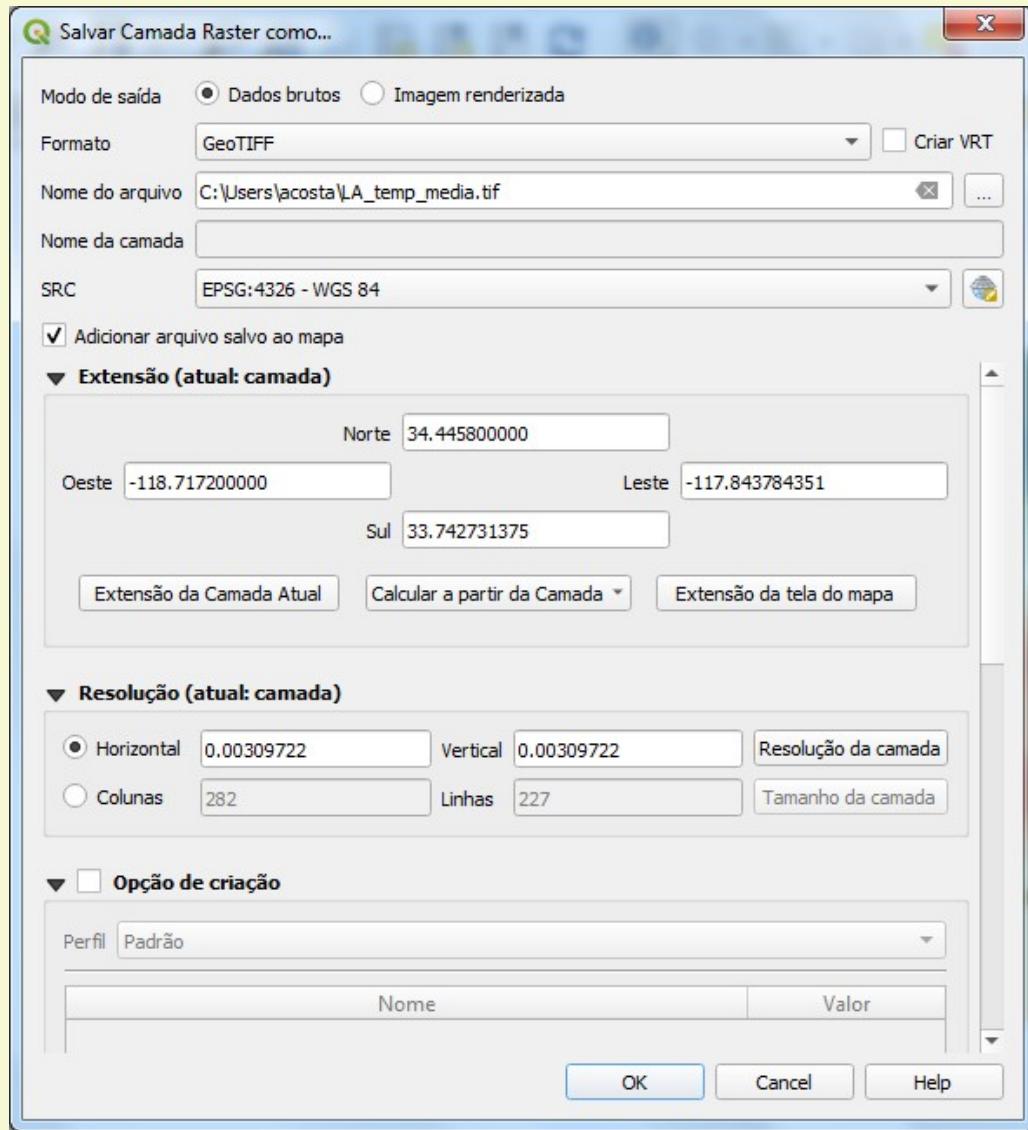
Ao executarmos o seguinte raster aparecerá na tela.



Vamos mover para baixo da camada 'temper' para podermos visualizar os pontos e vamos mudar a cor de 'cinza' para 'spectral invertido'. O resultado será:



Para gravar esse raster basta clicar com o botão direito do mouse na camada e selecionar **Exportar > Salvar como**. Escolha o nome e diretório e use os opções padrões apresentadas.



Pronto! Aqui finalizamos nossa breve introdução ao QGIS, muito mais pode ser feito mas o foco agora será o uso de Python no QGIS. Primeiros veremos o fundamentos da linguagem e depois cobriremos o seu uso no QGIS.

2. Fundamentos de Python

2.0 A linguagem Python - Introdução

Python foi criada por Guido Van Rossum em 1991 quando ele trabalhava para no Instituto de Pesquisa Nacional para Matemática e Ciência da Computação (CWI) na Holanda.

Em 2001 a linguagem passou a ser desenvolvida pela Python Software Foundation. Todo código, documentação e especificação, desde o lançamento da versão alfa 2.1, é propriedade da Python Software Foundation (PSF).

Instalação

QGIS já vem com python instalado e vamos usar o Console Python do QGIS inicialmente.

Console Python

Vamos utilizar o Console Python para aprender os conceitos básicos de python.

O console Python pode ser iniciado usando **CTRL+ALT+P** ou menu **Plugin > Python Console**:

```
1 Python Console
2 Use iface to access QGIS API interface or Type help(iface) for more info
3 Security warning: typing commands from an untrusted source can lead to data loss
and/or leak
```

Um exemplo ilustrativo de como criar um ponto (veremos a fundo mais adiante os detalhes):

```
>>> camada = QgsVectorLayer('Point?crs=epsg:4326','Manaus','memory')
>>> pr = camada.dataProvider()
>>> pt = QgsFeature()
>>> ponto1 = QgsPointXY(-60,-3.5)
>>> pt.setGeometry(QgsGeometry.fromPointXY(ponto1))
>>> pr.addFeatures([pt])
>>> camada.updateExtents()
>>> QgsProject.instance().addMapLayers([camada])
```

Ou carregando de um banco de dados e gravando como texto em arquivo local

```
>>> tabela = "rmp"
>>> geometria = "geom"
>>> uri = QgsDataSourceUri()
>>> uri.setConnection("amazeone.com.br","5432","dnpm","droid", "devcor")
>>> uri.setDataSource("public",tabela,geometria)
>>> processos=QgsVectorLayer(uri.uri(False),tabela,"postgres")
>>> QgsProject.instance().addMapLayer(processos)
>>> with open('C:/Users/voce/Desktop/cprm_ocorrencias.txt', 'w') as file:
...     for f in processos.getFeatures():
...         geom = f.geometry()
...         line='{{},{}},{{},{}},{{:.5f},{:.5f}}\n'.format(f['LABEL2'],
f['STATUS_ECO'],f['GRAU_DE_IM'],f['MUNICIPIO'],f['CLASSE_UTI'], geom.asPoint().y(),
geom.asPoint().x())
...         file.write(line)
```

2.1 Fundamentos da linguagem Python

Comentários de script se iniciam com # em python.

```
>>> # um comentário
>>> print('QGIS') # outro comentário
QGIS
>>> a ='# isso não é um comentário pois está entre aspas'
```

Tipos Numéricos

Python possui primitivamente os tipos numéricos de números inteiros (int), ponto flutuante (float) e complexo (complex).

```
>>> a = 1
>>> b = 3.2
>>> c = 4 + 3j
```

A função type (**variável**) retorna o tipo da variável.

```
>>> type(a)
<class 'int'>
>>> type(b)
<class 'float'>
>>> type(c)
<class 'complex'>
```

O resultado de uma divisão sempre será do tipo float.

```
>>> d = a/a
>>> d
1.0
>>> type(d)
<class 'float'>
```

O console pode funcionar como uma calculadora também

```
>>> 2+2
4
>>> 2**8 #dois elevado a oito
256
>>> 82%3 #resto inteiro da divisão
1
>>> 82//3 #quociente da divisão
27
>>> 3-2
1
>>> (50-5*6)/4 # Parênteses tem precedente na operação
5.0
```

Tipo Alfanumérico

Em python o tipo str é usado para palavras, textos e caracteres alfanuméricos.

```
>>> nome = 'Manuel'
>>> type(nome)
<class 'str'>
>>> letra = 'a'
>>> type(letra)
<class 'str'>
```

Um objeto da classe str nada mais é do que uma matriz de valores sequenciados onde o primeiro valor (caractere) corresponde ao índice 0 da matriz e o último valor ao índice -1, o penúltimo -2 e assim sucessivamente.

```
>>> nome[0]
'M'
>>> nome[1]
'a'
>>> nome[-1]
'l'
>>> nome[-2]
'e'
>>> nome[-6]
'M'
>>> nome[2:5] #note que o valor do índice 5 não está incluído
'nue'
```

Um objeto str uma vez definido é imutável e se tentarmos mudar o valor de um objeto str uma mensagem de erro aparecerá.

```
>>> nome[2]='t'
Traceback (most recent call last):
  File "/usr/lib/python3.7/code.py", line 90, in runcode
    exec(code, self.locals)
  File "<input>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

A função len() retorna o comprimento do objeto str.

```
>>> palavra='anticonstitucionalíssimamente'
>>> len(palavra)
29
```

Podemos usar aspas simples ou duplas para delimitar um objeto str para podermos definir estas como:

```
>>> s1 = "Bom dia senhor O'Brien"
>>> print(s1)
Bom dia senhor O'Brien
>>> s2= 'Quoth the raven "Nevermore."'
>>> print(s2)
Quoth the raven "Nevermore."
```

Tipo lista

Python possui diversos tipos compostos de dados, a lista (list) e uma delas.

Uma lista é um conjunto de objetos de determinado tipo ou de tipos distintos armazenados em uma lista.

Listas são declaradas dentro de colchetes onde cada objeto (item) dela é separado por vírgula.

```
>>> lista = [3200,2670,3100,3000]
>>> lista2 = ['gelo', 'limão', 'açúcar', 51,15.99]
```

Assim como str, cada item de uma lista pode ser acessado usando índices.

```
>>> lista[0]
3200
>>> lista[1]
2670
>>> lista[3]
3000
>>> lista[1:3]
```

```
[2670, 3100]
>>> type(lista2[0])
<class 'str'>
>>> type(lista2[3])
<class 'int'>
>>> type(lista2[4])
<class 'float'>
>>> lista2[-1]
15.99
```

Ao contrário do objeto str, os valores de itens de uma lista podem ser modificados

```
>>> lista2[3]='pinga'
>>> lista2
['gelo', 'limão', 'açúcar', 'pinga', 15.99]
```

Uma lista pode conter outras listas como itens. E acessamos cada item dessa lista interna usando um índice adicional.

```
>>> lista2[1]=['abacaxi','maracujá','limão']
>>> lista2
['gelo', ['abacaxi', 'maracujá', 'limão'], 'açúcar', 'pinga', 15.99]
>>> lista2[1][1]
'maracujá'
>>> lista2[1][-1]
'limão'
```

Podemos adicionar itens a uma lista já existente usando adição ou a função append().

```
>>> lista2 = lista2 + ['guardanapo', 'canudo']
>>> lista2.append('copo')
>>> lista2
['gelo', ['abacaxi', 'maracujá', 'limão'], 'açúcar', 'pinga', 15.99, 'guardanapo',
'canudo', 'copo']
```

Alguns métodos de interação com listas são mostradas abaixo:

Criamos a seguinte lista vazia inicialmente.

```
>>> lis=[]
>>> lis
[]
```

Adicionando itens na lista com o método extend().

```
>>> lis.extend([1,2,3])
>>> lis
[1, 2, 3]
```

Adicionando um item de valor 10 na posição predeterminada (0) com o método insert().

```
>>> lis.insert(0,10)
>>> lis
[10, 1, 2, 3]
```

Removendo da lista a primeira ocorrência do valor passado pelo método remove().

```
>>> lis.remove(2)
>>> lis
[10, 1, 3]
```

Podemos copiar uma lista usando o método copy().

```
>>> lis2=lis.copy()
>>> lis2
```

```
[10, 1, 3]
```

Revertemos a ordem dos itens de uma lista com o método reverse().

```
>>> lis2.reverse()  
>>> lis2  
[3, 1, 10]
```

Ordenamos uma lista usando o método sort().

```
>>> lis2.sort()  
>>> lis2  
[1, 3, 10]
```

O método pop() remove e retorna o item indicado pelo índice dado, se nenhum índice é fornecido o último item é removido da lista e retornado.

```
>>> lis3 = [1,1,2,3,4,4,4,3,2,3,1]  
>>> lis3.pop(1)  
1  
>>> lis3  
[1, 2, 3, 4, 4, 4, 3, 2, 3, 1]
```

O método index(x[,início[,fim]]) retorna o índice (na base 0) da primeira ocorrência do item x. O segundo argumento mostra a partir de qual e até qual índice da lista procurar. Caso não encontre um item com o valor uma mensagem de erro é retornada.

```
>>> lis3.index(1,1,10)  
9  
>>> lis3.index(1)  
0  
>>> lis3.index(1,1)  
9  
>>> lis3.index(1,1,10)  
9  
>>> lis3.index(8,1,10)  
Traceback (most recent call last):  
  File "C:/PROGRA~1/QGIS3~1.4/apps/Python37/lib/code.py", line 90, in runcode  
    exec(code, self.locals)  
  File "<input>", line 1, in <module>  
ValueError: 8 is not in list
```

O método count(x) retorna o número de vezes que o item x aparece na lista.

```
>>> lis3.count(4)  
3
```

O método clear() remove todos os itens da lista.

```
>>> lis3.clear()  
>>> lis3  
[]
```

Podemos usar a instrução del para deletar itens de uma lista usando índices em vez de valores.

```
>>> lis3 = [1,1,2,3,4,4,4,3,2,3,1]  
>>> del lis3[2]  
>>> lis3  
[1, 1, 3, 4, 4, 4, 3, 2, 3, 1]  
>>> del lis3[5:8]  
>>> lis3  
[1, 1, 3, 4, 4, 3, 1]
```

Tuples

Assim como str e listas, tuples são dados em sequência usados em python. As diferenças principais entre uma tuple e uma lista é que tuples são declaradas usando vírgulas para separar os itens e esses itens são imutáveis.

```
>>> t = 'banana', 3,45, False, "oi!"  
>>> t  
('banana', 3, 45, False, 'oi!')  
>>> t[0]  
'banana'  
>>> doist =t,(1,2,3,4,5)  
>>> doist  
(('banana', 3, 45, False, 'oi!'), (1, 2, 3, 4, 5))  
>>> doist[0]  

```

Sets

Sets ou conjuntos são outro tipo de dados em sequência que python utiliza. Sets são definidos dentro de chaves {} e resultam em uma simples aparição de cada um de seus itens e estes não são indexados.

```
>>> conjunto = {'rio','terra','fogo','fogo','água','terra'}  
>>> conjunto  
{'terra', 'fogo', 'rio', 'água'}  
>>> 'rio' in conjunto  
True  

```

Dicionários

Dicionários são um tipo de dados bastante usado em python. Um dicionário possui sempre uma chave e um valor, esta chave é usada no lugar de um índice numérico que é usado numa lista. Essa chave deve ser única e um dicionário vazio pode ser criado usando {}.

```
>>> dicio={'nome':'andre','sobrenome':'costa'}  
>>> dicio['idade']=51  
>>> dicio  
{'nome': 'andre', 'sobrenome': 'costa', 'idade': 51}  
>>> list(dicio)
```

```

['nome', 'sobrenome', 'idade']
>>> sorted(dicio)
['idade', 'nome', 'sobrenome']
>>> dicio[1]
Traceback (most recent call last):
  File "C:/PROGRA~1/QGIS3~1.4/apps/Python37/lib/code.py", line 90, in runcode
    exec(code, self.locals)
  File "<input>", line 1, in <module>
KeyError: 1
>>> del dicio['idade']
>>> dicio
{'nome': 'andre', 'sobrenome': 'costa'}
>>> dicio2=dict([('código', 34), ('senha', 65483), ('acessos', 8)])
>>> dicio2
{'código': 34, 'senha': 65483, 'acessos': 8}

```

As seguintes funções internas são usadas para conversão de tipos e informações de alguns tipos.

```

>>> f=-5.789
>>> round(f,2)
-5.79
>>> abs(f)
5.789
>>> int(f)
-5
>>> str(f)
'-5.789'
>>> lis=[2,3,45,12,78,1,-17,3]
>>> min(lis)
-17
>>> max(lis)
78
>>> sum(lis)
127
>>> sorted(lis)
[-17, 1, 2, 3, 3, 12, 45, 78]
>>> i=255
>>> hex(i)
'0xff'
>>> bin(i)
'0b11111111'
>>> float(i)
255.0
>>> chr(i)
'\u00f3'
>>> ord('\u00f3')
255
>>> oct(i)
'0o377'
>>> ascii(i)
'255'
>>> pow(2,10)
1024
>>> bool(1<2 and 3>2)
True
>>> bool(1<2 and 3>4)
False
>>> bool(1<2 or 3>4)
True

```

As seguintes palavras são reservadas da linguagem python e não podem ser usadas para definir variáveis.

```
and      except      lambda      with
as       finally     None        while
assert   False       for         yield
break    from        not         or
class    global      pass        raise
continue if          return
def      import
elif    in          True
else    is          try
```

E essas são as funções internas:

abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

2.2 Controles de fluxo

Como toda linguagem de programação, python utiliza controles de fluxo de programa. Mas antes disso vamos falar um pouco de indentação de script.

Obrigatoriamente em python temos de usar indentação de blocos de código uma vez que não os separamos por chaves {}, assim todo bloco de código, seja ele uma classe, função ou um controle de fluxo, deve ser indentado. Num bloco com indentação no console, o >>> se transforma em ... indicando que estamos dentro de um bloco no script. Quando todas as instruções do bloco estão finalizadas, teclamos ‘enter’ na linha final com ...

if elif else

O if talvez seja a instrução mais conhecida em programação. Ela checa se (if) uma condição ou se outras condições (elif) são atendidas. Se nenhuma condição for atendida, podemos também instruir que algo seja feito (else).

```
>>> x = int(input("Diga um número inteiro: "))
Diga um número inteiro: 2
>>> if x<0:
...     print('O número é negativo')
... elif x>0:
...     print('O número é positivo')
... else:
```

```
...     print('O número é zero')
...
O número é positivo
```

for

A repetição (loop) `for` é definida como “executar/repetir as instruções nos termos predefinidos”. No exemplo abaixo a repetição é feita para cada palavra da variável `palavras` e a palavra e o comprimento dela é impresso como resultado.

```
>>> palavras = ['Olá!', 'Vamos', 'aprender', 'python?']
>>> for palavra in palavras:
...     print(palavra, len(palavra))
...
Olá! 4
Vamos 5
aprender 8
python? 7
```

while

A repetição `while` é definida com “enquanto o que foi predefinido não ocorrer, vai executando/repetindo”.

```
>>> a, b = 0, 1
>>> while a < 1000:
...     print(a, end=', ')
...     a, b = b, a+b
...
0,1,1,2,3,5,8,13,21,34,55,89,144,233,377,610,987,
```

range()

A forma que python interage com uma série numérica é usando a função `range()`.

```
>>> for i in range(10):
...     print(i, end=', ')
...
0,1,2,3,4,5,6,7,8,9,
```

Vamos supor que não sabemos a quantidade de itens numa lista mas queremos interagir (no caso listar) cada um destes itens. Usamos `range` para nos auxiliar.

```
>>> palavras = ['Olá!', 'Vamos', 'aprender', 'python?']
>>> for i in range(len(palavras)):
...     print(palavras[i], i)
...
Olá! 0
Vamos 1
aprender 2
python? 3
```

Podemos definir o início e final de `range()` e o passo também.

```
>>> list(range(50, 57))
[50, 51, 52, 53, 54, 55, 56]
>>> list(range(10, 5, -1))
[10, 9, 8, 7, 6]
>>> list(range(-100, -50, 10))
[-100, -90, -80, -70, -60]
```

Break, continue e else em loops

A instrução `break` quebra a execução da repetição `for` ou `while` mais interna.

Repetições também podem ter uma instrução `else`; ela é executada quando a repetição termina mas não quando ela é interrompida por uma instrução `break`. Veja o exemplo abaixo:

```
>>> for n in range(2,10):
...     for x in range(2, n):
...         if n % x == 0:
...             print(n, 'é igual a ', x, '*', n//x)
...             break
...     else:
...         print(n, 'é um número primo')
...
2 é um número primo
3 é um número primo
4 é igual a 2 * 2
5 é um número primo
6 é igual a 2 * 3
7 é um número primo
8 é igual a 2 * 4
9 é igual a 3 * 3
```

A instrução `continue`, avança para a próxima interação da repetição:

```
>>> for num in range(2, 9):
...     if num % 2 == 0:
...         print("Número par", num)
...         continue
...     print("Número ímpar", num)
...
Número par 2
Número ímpar 3
Número par 4
Número ímpar 5
Número par 6
Número ímpar 7
Número par 8
```

2.3 Funções

Uma função é um conjunto de instruções organizados e reusáveis para executar uma tarefa. Em python definimos uma função usando `def` seguido do nome da função e dos argumentos ou parâmetros passados dentro de parênteses. Veja o exemplo abaixo:

```
>>> def fibo(n):
...     """Calcula a série de Fibonacci até o limite informado"""
...     a,b=0,1
...     while a<n:
...         print(a,end=' ')
...         a,b=b,a+b
...     print()
...
>>> fibo(100)
0 1 1 2 3 5 8 13 21 34 55 89
>>> fibo(1000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
>>> fibo(10000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765
```

Uma outra forma de escrever essa função, utilizando um valor a ser retornado desta no formato de lista, é mostrada a seguir:

```
>>> def fibo2(n):
...     """Calcula e retorna a série de Fibonacci até o limite informado"""
...     resultado=[]
...     a,b=0,1
...     while a<n:
...         resultado.append(a)
...         a,b=b,a+b
...     return resultado
...
>>> fibo2(100)
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

Podemos assinalar uma função a uma variável.

```
>>> f=fibo2
>>> f(10)
[0, 1, 1, 2, 3, 5, 8]
```

Se passarmos nossa função como argumento para a função `help` teremos a string de documentação (entre as aspas duplas repetidas três vezes) como informação retornada. Essa é uma maneira de documentar o que uma função faz em python.

```
>>> help(fibo2)
Help on function fibo2 in module __main__:
```

```
fibo2(n)
    Calcula e retorna a série de Fibonacci até o limite informado
```

É possível definir funções com número de argumentos variáveis.

-Valor padrão predefinido é a forma mais comum onde podemos predefinir o valor de um ou mais parâmetros de uma função.

```
>>> def pergunta(texto, tentativas=4, msg='Tente de novo!'):
```

```

...
    while True:
...
        ok = input(texto)
...
        if ok in ('Sim', 'sim', 's', 'S'):
            return True
...
        if ok in ('n', 'não', 'N', 'Não', 'nao', 'Nao'):
            return False
...
        tentativas = tentativas - 1
...
        if tentativas < 0:
            raise ValueError('Resposta Inválida.')
...
        print(msg)
...
>>> pergunta('Esta com fome?')
Esta com fome?j
Tente de novo!
Esta com fome?Sim
True
>>> pergunta('Está com fome?', 1, 'Não entendi a resposta!')
Está com fome?iop
Não entendi a resposta!
Está com fome?poi
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 10, in pergunta
ValueError: Resposta Inválida.

```

Vamos ver agora um exemplo prático de como trabalhar com funções criando um nosso primeiro script. Crie o arquivo **temp_converter.py** com o seguinte código.

```

#!/usr/bin/env python3
'''Converte temperaturas de Celsius para Fahrenheit, Kelvin para Celsius e Kelvin para Fahrenheit.
Uso:
    Carregue usando import
        import temp_converter as tc
Autor:
    Seu Nome - 03.12.2019'''
def celsius_para_fahr(temp_celsius):
    return 9/5 * temp_celsius + 32
def kelvins_para_celsius(temp_kelvins):
    return temp_kelvins - 273.15
def kelvins_para_fahr(temp_kelvins):
    temp_celsius = kelvins_para_celsius(temp_kelvins)
    temp_fahr = celsius_para_fahr(temp_celsius)
    return temp_fahr

```

Agora vamos importar nosso script e usar funções dele.

```

>>> import temp_converter as tc
>>> print("O ponto de congelamento da água em Fahrenheit é:",
tc.celsius_para_fahrenheit(0))
O ponto de congelamento da água em Fahrenheit é: 32.0
>>> print('O Zero absoluto em Fahrenheit é:', tc.kelvins_para_fahr(0))
O Zero absoluto em Fahrenheit é: -459.6699999999996

```

2.4 Módulos

Em python um module (módulo) é simplesmente um arquivo com extensão .py com classes, funções e demais instruções. Nosso script acima é um exemplo de um módulo. Um package (pacote) é uma forma de organizar vários módulos em uma entidade maior.

Algumas linguagem chamam módulos e pacotes de library. Um module é carregado usando o comando `import` e podemos renomear um módulo usando as

```
>>> import math as m  
>>> m.sqrt(81)  
9
```

Também podemos importar uma simples função de um módulo usando `from`

```
>>> from math import sqrt  
>>> sqrt(9)  
3
```

Ou podemos importar um submódulo de um módulo.

```
>>> import matplotlib.pyplot as plt  
>>> plt.figure()  
<Figure size 432x288 with 0 Axes>
```

Usaremos o tempo todo vários módulos, essa é e força da linguagem python com inúmeros módulos existentes para as mais diversas funções. Vamos iniciar vendo o módulo pandas.

2.5 Pandas

A biblioteca (library) pandas foi desenvolvida por Wes McKinney como uma alternativa a linguagem R para lidar com estrutura de dados mais complexas. Hoje é uma library potente e moderna largamente utilizadas em diversas áreas da ciência.

Panda tira vantagem em utilizar outra library chamada numpy escrita em C e portanto, bastante rápida e eficiente ao lidar com dados em grandes volumes. Os seguintes formatos podem ser importados e exportados usando pandas:

Formato	Tipo de dado	Le	Escreve
texto	CSV	read_csv	to_csv
texto	JSON	read_json	to_json
texto	HTML	read_html	to_html
texto	Local clipboard	read_clipboard	to_clipboard
binário	MS Excel	read_excel	to_excel
binário	HDF5 Format	read_hdf	to_hdf
binário	Feather Format	read_feather	to_feather
binário	Msgpack	read_msgpack	to_msgpack
binário	Stata	read_stata	to_stata
binário	SAS	read_sas	
binário	Python Pickle Format	read_pickle	to_pickle
SQL	SQL	read_sql	to_sql
SQL	Google Big Query	read_gbq	to_gbq

Baixe o arquivo **curvelo.csv** e inicie o python. Vamos trabalhar com pandas lendo o conteúdo desse arquivo.

```
>>> import pandas as pd
>>> data = pd.read_csv('curvelo.csv')
>>> data.head()
   codigo_estacao data hora ... vento_rajada radiacao precipitacao
0      A538  01/01/2019    16 ...     6.7    2749.00        0.0
1      A538  01/01/2019     8 ...     3.8     -2.63        0.2
2      A538  01/01/2019    11 ...     1.8     977.30        0.0
3      A538  01/01/2019    12 ...     2.6    1349.00        0.0
4      A538  01/01/2019    15 ...     6.3    3561.00        0.0

[5 rows x 20 columns]
>>> type(data)
<class 'pandas.core.frame.DataFrame'>
>>> len(data)
7545
>>> data.shape
(7545, 20)
>>> data.columns.values
array(['codigo_estacao', 'data', 'hora', 'temp_inst', 'temp_max',
       'temp_min', 'umid_inst', 'umid_max', 'umid_min',
       'pto_orvalho_inst', 'pto_orvalho_max', 'pto_orvalho_min',
       'pressao', 'pressao_max', 'pressao_min', 'vento_direcao',
       'vento_vel', 'vento_rajada', 'radiacao', 'precipitacao'],
      dtype=object)
>>> data.dtypes
codigo_estacao      object
data                  object
hora                 int64
temp_inst            float64
temp_max             float64
temp_min             float64
umid_inst            int64
umid_max             float64
umid_min             float64
pto_orvalho_inst    float64
pto_orvalho_max     float64
pto_orvalho_min     float64
pressao              float64
pressao_max          float64
pressao_min          float64
vento_direcao        float64
vento_vel             int64
vento_rajada         float64
radiacao             float64
precipitacao         float64
dtype: object
```

Podemos selecionar colunas de dados da seguinte forma:

```
>>> selecao=data[['pressao_min','pressao_max']]
>>> selecao.head()
   pressao_min  pressao_max
0      940.7      941.1
1      939.3      940.1
2      941.3      941.9
3      941.8      942.2
4      941.1      941.6
```

Aplicando estatística descritiva nos dados

```
>>> selecao.mean()
pressao_min    940.133461
pressao_max    940.648111
dtype: float64
>>> selecao.max()
pressao_min    950.9
pressao_max    951.3
dtype: float64
>>> selecao.min()
pressao_min    931.3
pressao_max    931.8
dtype: float64
>>> selecao.median()
pressao_min    939.7
pressao_max    940.2
dtype: float64
>>> selecao.std()
pressao_min    3.248884
pressao_max    3.221912
dtype: float64
>>> selecao.describe()
           pressao_min   pressao_max
count    7543.000000  7543.000000
mean     940.133461   940.648111
std      3.248884    3.221912
min     931.300000  931.800000
25%    937.900000  938.500000
50%    939.700000  940.200000
75%    942.100000  942.600000
max     950.900000  951.300000
```

Vamos agora usar pandas para fazer o caminho inverso , de lista de dados para arquivo **estacoes.csv**.

```
>>> estacao=['E-01','E-02','E-03','E-04','E-05','E-06']
>>> latitude=[-3.34,-3.23,-3.12,-3.32,-3.33,-3.19]
>>> longitude=[-60.12,-60.43,-60.11,-60.54,-59.87,-60.00]
>>> dadoEst = pd.DataFrame(data = {"Estação" : estacao, "latitude" : latitude,
"longitude" : longitude})
>>> dadoEst
   Estação  latitude  longitude
0      E-01     -3.34     -60.12
1      E-02     -3.23     -60.43
2      E-03     -3.12     -60.11
3      E-04     -3.32     -60.54
4      E-05     -3.33     -59.87
5      E-06     -3.19     -60.00
>>> dadoEst.to_csv('estacoes.csv')
```

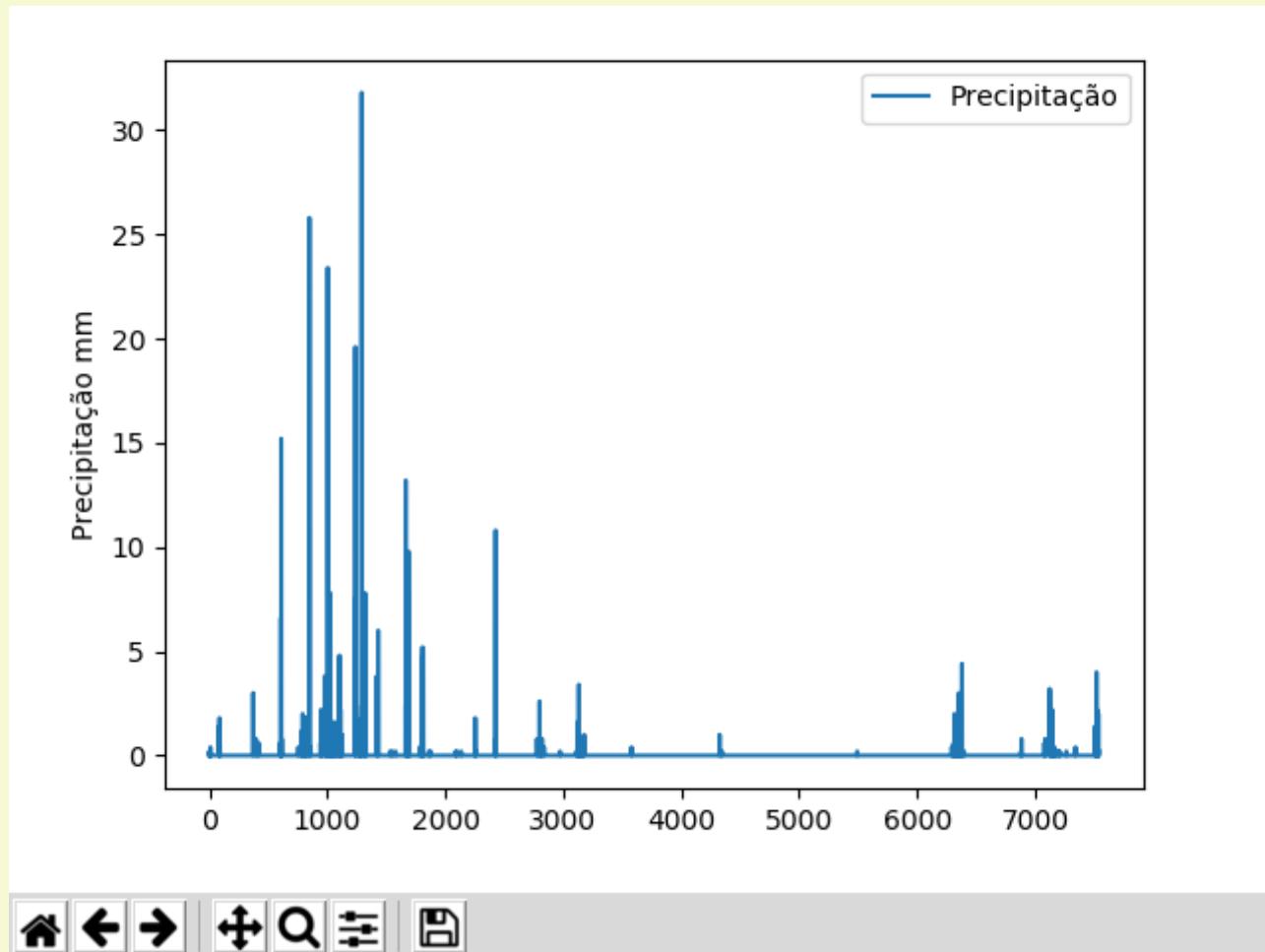
Por último, criando um data frame vazio.

```
>>> df = pd.DataFrame()
>>> print(df)
Empty DataFrame
Columns: []
Index: []
```

2.6 Gráficos

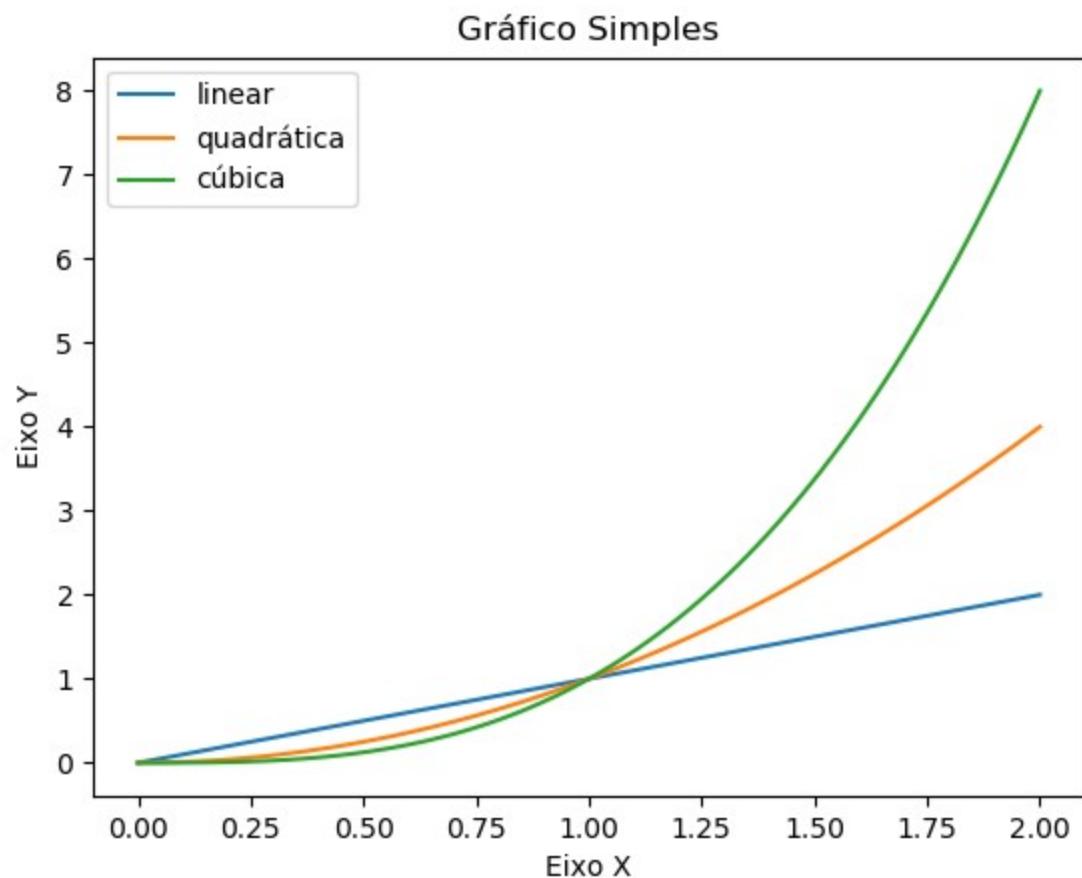
Vamos mostrar simplificadamente como podemos criar gráficos com python. Existem diversas libraries para a criação de gráficos mas aqui vamos usar o matplotlib com o auxílio de pandas para criar alguns gráficos básicos.

```
>>> import pandas as pd
>>> import matplotlib.pyplot as plt
>>> data = pd.read_csv('curvelo.csv')
>>> plt.plot(data[['precipitacao']], label='Precipitação')
[<matplotlib.lines.Line2D object at 0x7f55bbc70d90>]
>>> plt.ylabel('Precipitação mm')
Text(0, 0.5, 'Precipitação mm')
>>> plt.legend()
<matplotlib.legend.Legend object at 0x7f55bcee2b50>
>>> plt.show()
```



Vamos usar o numpy para criar uma série de números entre 0 e 2 e plotar a sequência linear, quadrática e cúbica de série para ilustrar como criar um gráfico com mais de uma curva.

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> x = np.linspace(0, 2, 100)
>>> plt.plot(x, x, label='linear')
>>> plt.plot(x, x**2, label='quadrática')
>>> plt.plot(x, x**3, label='cúbica')
>>> plt.ylabel('Eixo Y')
>>> plt.xlabel('Eixo X')
>>> plt.title("Gráfico Simples")
>>> plt.legend()
>>> plt.show()
```



3. Usos do Python no QGIS

Vamos introduzir as classes do PyQGIS na medida que avançamos nos pontos cobertos. Uma classe em python é a **definição** de um tipo de objeto e de métodos (funções) associados a este objeto. Por exemplo um projeto, uma camada raster, uma camada vetorial, etc.

3.1 Primeiros passos, noções de Classes

A library PyQGIS é bastante extensa com diversas classes. Vamos aqui cobrir as classes mais básicas e essenciais para podermos daí ter uma boa base para desenvolver scripts mais complexos.

Classe Projeto (QgsProject) – Criar e ler um Projeto

Um projeto armazena um conjunto de informações sobre camadas, estilos, layouts, anotações etc. Como se trata de uma classe singleton, criamos um novo objeto usando o método `QgsProject.instance()`. Vamos mostrar como criar um projeto vazio chamado `meu_projeto.qgs` usando o método `write()`.

```
>>> projeto= QgsProject.instance()
>>> projeto.write('c:/users/voce/meu_projeto.qgs')
>>> print(projeto.fileName())
c:/users/voce/meu_projeto.qgs
```

Agora vamos sair do QGIS e entrar novamente para carregarmos o projeto que criamos usando python.

```
>>> projeto=QgsProject.instance()
>>> projeto.read('c:/users/voce/meu_projeto.qgs')
>>> print(projeto.fileName())
c:/users/voce/meu_projeto.qgs
```

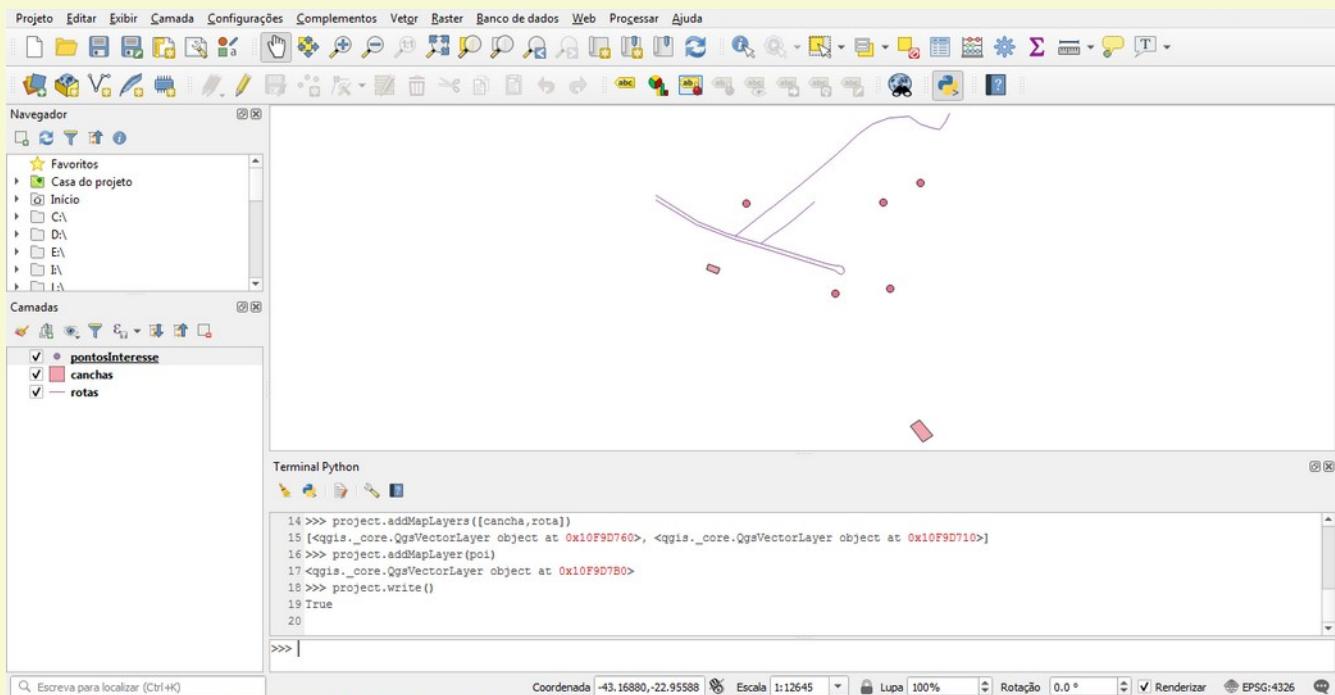
Na medida que formos vendo as outras classes, vamos ver outros métodos associados à classe Projeto.

Classe Vetor (QgsVectorLayer) – Adicionar Camada Vetorial

Um objeto do tipo camada vetorial é usado para carregarmos e interagirmos com camadas do tipo vetor. Vamos carregar o nosso projeto e adicionar nele três camadas vetor criadas anteriormente usando os métodos de projeto `addMapLayer` e `addMapLayers`. Criamos um objeto de classe camada vetorial usando o método `QgsVectorLayer()` passando o caminho para o arquivo vetorial, o identificador que nossa camada terá, e a biblioteca a ser usada (ogr nesse caso). Por último salvamos o nosso projeto com o método `write()`.

```
>>> projeto=QgsProject.instance()
>>> projeto.read('c:/users/voce/meu_projeto.qgs')
>>> cancha=QgsVectorLayer("c:/users/voce/cancha.shp","canchas", "ogr")
>>> rota=QgsVectorLayer("c:/users/voce/rota.shp","rotas", "ogr")
>>> poi=QgsVectorLayer("c:/users/voce/poi.shp","pontosInteresse", "ogr")
>>> projeto.addMapLayers([cancha,rota])
>>> projeto.addMapLayer(poi)
>>> projeto.write()
```

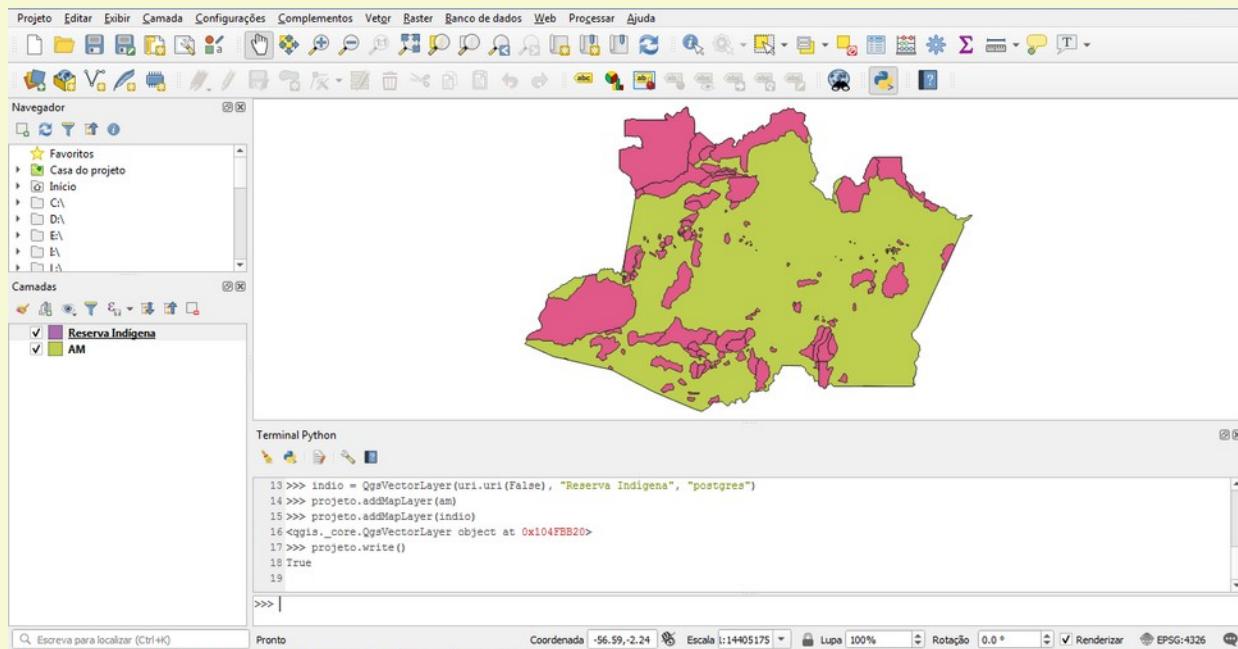
Algo similar ao apresentado abaixo deverá aparecer:



Agora vamos fazer uso de uma outra classe para podermos carregar uma camada vetorial localizada em um banco de dados Postgis remoto. A classe é a `QgsDataSourceUri` e usaremos os métodos `setConnection()` e `setDataSource()` para extrairmos uma tabela espacial vetorial. Criaremos um projeto novo, adicionaremos uma camada local e uma camada remota Postgis e por último vamos gravar o projeto.

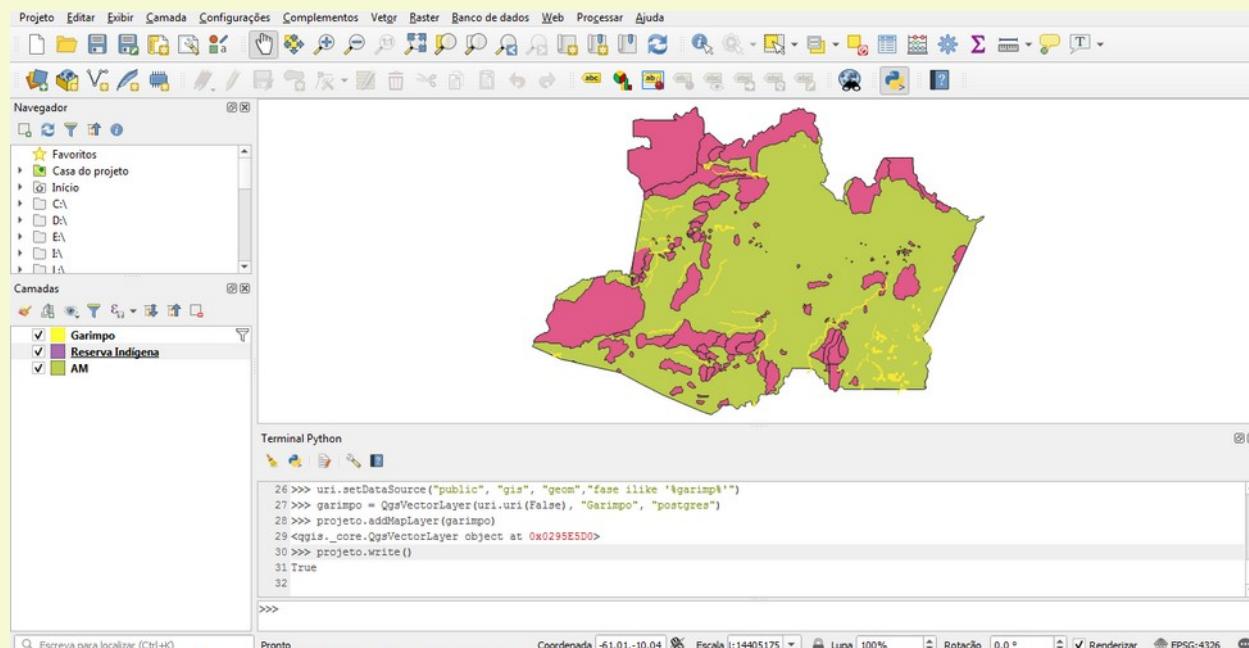
```
>>> projeto = QgsProject.instance()
>>> projeto.write('c:/users/voce/meu_projeto2.qgs')
>>> am=QgsVectorLayer("c:/users/voce/amazonas.shp", "AM", "ogr")
>>> projeto.addMapLayer(am)
>>> uri = QgsDataSourceUri()
>>> uri.setConnection("amazeone.com.br", "5432", "dnpm", "droid", "devcor")
>>> uri.setDataSource("public", "indio", "geom")
>>> indio = QgsVectorLayer(uri.uri(False), "Reserva Indígena", "postgres")
>>> projeto.addMapLayer(indio)
>>> projeto.write()
```

Um projeto conforme o ilustrado abaixo deverá aparecer.



O método `setConnection()` tem como parâmetros o endereço do servidor (IP ou DNS), a porta (geralmente 5432), o banco de dados, o usuário e a senha. O método `setDataSource()` tem como parâmetros o esquema da tabela, o nome da tabela e a coluna com o elemento geométrico espacial. Alternativamente podemos adicionar uma cláusula SQL WHERE como o quarto argumento. Vamos ver um exemplo onde extraímos uma camada vetorial somente os requerimentos de garimpo e permissão de lavra garimpeira dos requerimentos do estado do Amazonas e adicionamos ela no nosso projeto já criado acima.

```
>>> uri.setDataSource("public", "gis", "geom", "fase ilike '%garimp%'")
>>> garimpo = QgsVectorLayer(uri.uri(False), "Garimpo", "postgres")
>>> projeto.addMapLayer(garimpo)
>>> projeto.write()
```



Antes de movermos para o próximo tópico vamos dar uma olhada em alguns métodos da Classe Projeto (QsgProject) relacionados a classe Camadas Vetoriais (QsgVectorLayer).

count retorna o número de camadas válidas do projeto.

```
>>> mprojeto.count()  
3
```

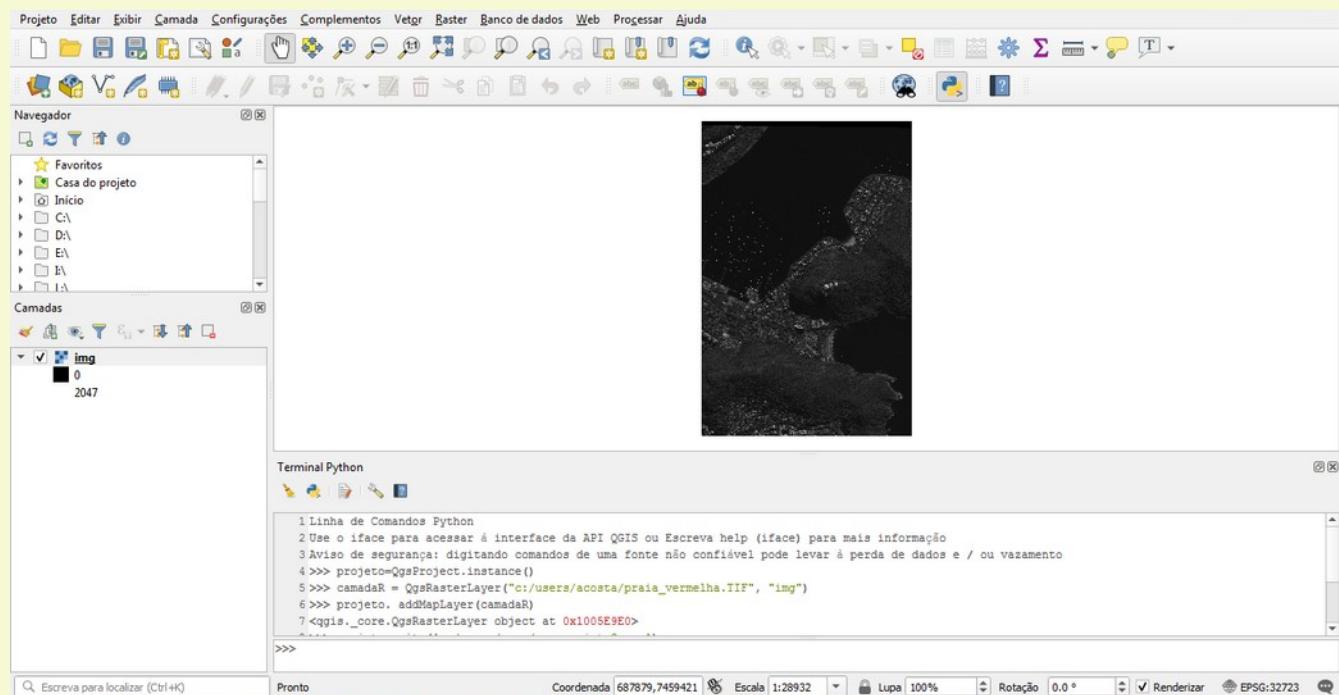
mapLayers retorna um mapa das camadas existentes do projeto.

```
>>> projeto.mapLayers()  
{'AM_ff8ab0e0_06d4_4a4b_9207_303720e92dfa': <qgis._core.QgsVectorLayer object at  
0x104FBA80>, 'Garimpo_6f114cc3_f5e7_46eb_9f80_ff9262d36ede':  
<qgis._core.QgsVectorLayer object at 0x0295E5D0>,  
'Reserva_Indigena_de15cc66_19d3_4a36_9633_87b891921638': <qgis._core.QgsVectorLayer  
object at 0x104FBB20>}
```

Classe Raster (QgsRasterLayer) - Adicionar Camada Raster

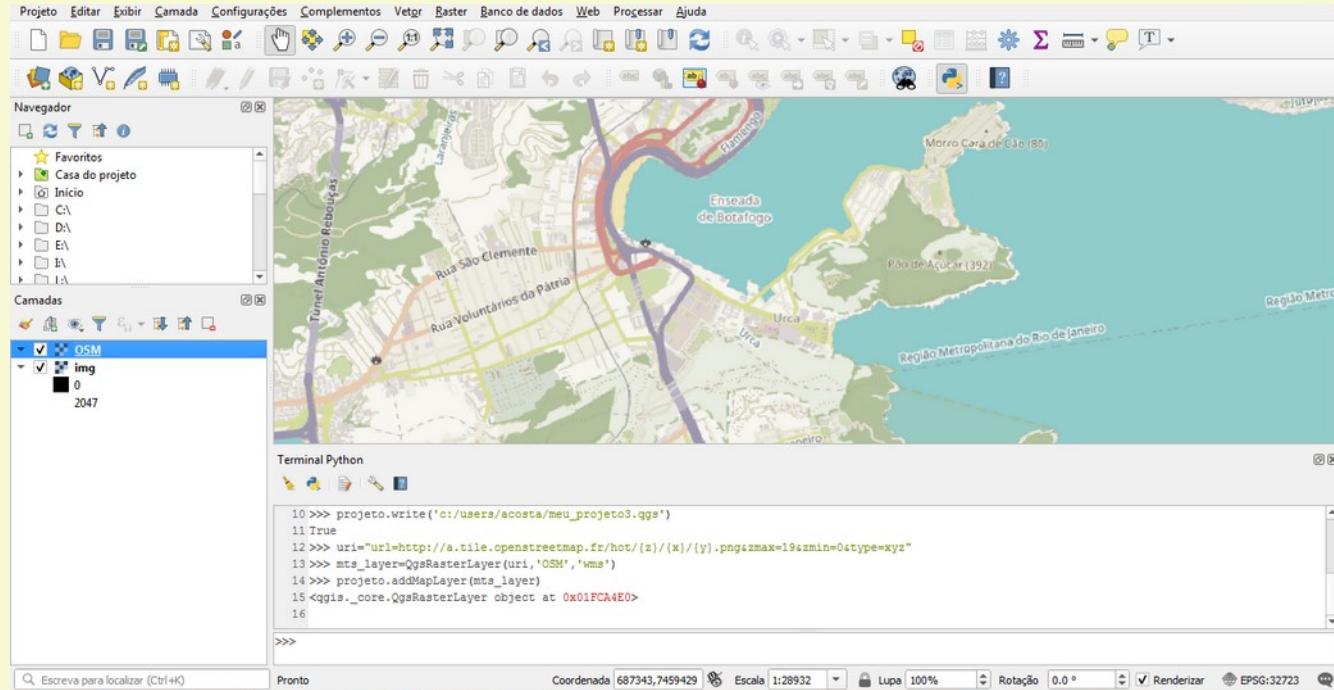
Similar à forma que adicionamos camadas vetoriais, podemos adicionar imagens raster no nosso projeto usando objeto da classe raster. Vamos criar um projeto e adicionar uma imagem raster nele.

```
>>> projeto=QgsProject.instance()  
>>> camadaR = QgsRasterLayer("c:/users/voce/praias_vermelha.TIF", "img")  
>>> projeto.addMapLayer(camadaR)  
>>> projeto.write('c:/users/voce/meu_projeto3.qgs')
```



Podemos também adicionar dados do tipo raster usando provedores do tipo TMS (TileMapService) ou WMS (WebMapService).

```
>>> uri="url=http://a.tile.openstreetmap.fr/hot/{z}/{x}/{y}.png&zmax=19&zmin=0&type=xyz"
>>> mts_layer=QgsRasterLayer(uri,'OSM','wms')
>>> projeto.addMapLayer(mts_layer)
>>> projeto.write()
```



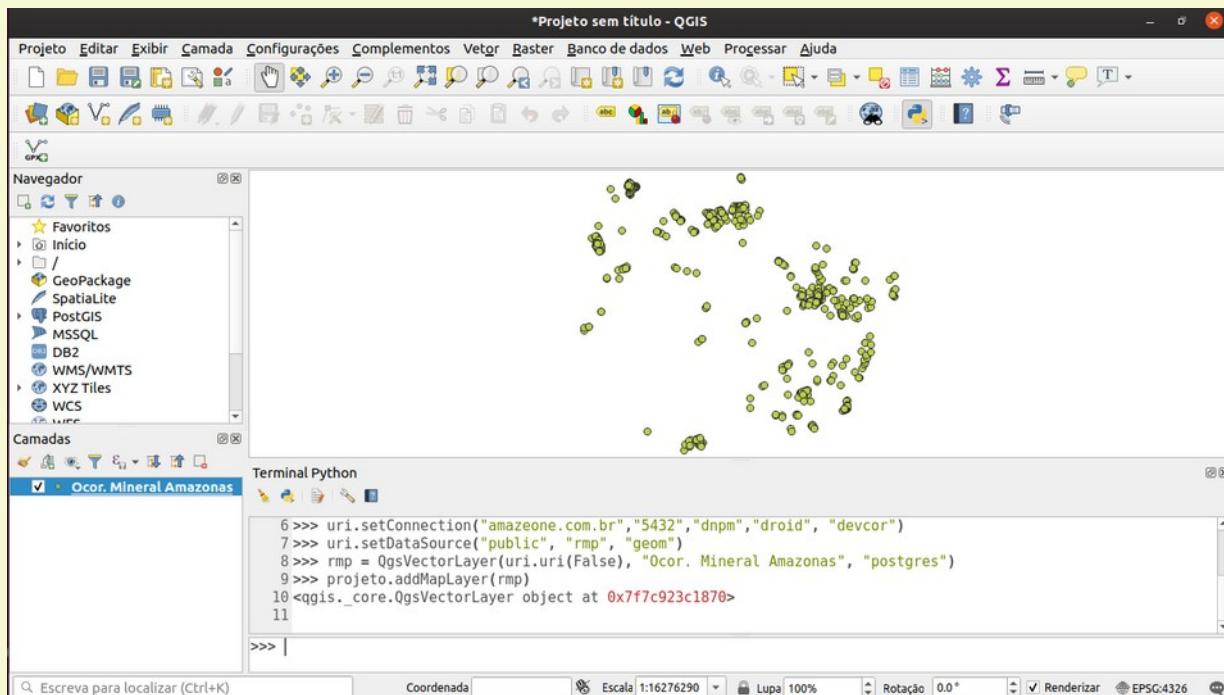
3.2 Interagindo com informações de objetos da classe Vector

Podemos obter diversas informações sobre objetos vetoriais tais como, projeções, extensão, número de elementos, valores e nomes dos campos de atributos (colunas) e até criar um metadata da camada (com a informação existente).

Vamos primeiro carregar um dado do tipo ponto de um banco Postgis remoto.

```
>>> projeto = QgsProject.instance()
>>> uri = QgsDataSourceUri()
>>> uri.setConnection("amazeone.com.br","5432","dnpm","droid", "devcor")
>>> uri.setDataSource("public", "rmp", "geom")
>>> rmp = QgsVectorLayer(uri.uri(False), "Ocor. Mineral Amazonas", "postgres")
>>> projeto.addMapLayer(rmp)
```

A camada será carregada conforme a ilustração mostrada abaixo. Vamos agora acessar as informações que mais usadas de maneira geral. Outras informações existem no objeto camada, veja a documentação para mais detalhes. Vamos nos ater às informações e dados mais usados.



O sistema de referência de coordenadas CRS (Coordinate Reference System)

O método crs() retorna o sistema de referência de coordenada original do objeto camada que o invoca.

```

>>> crs=rmp.crs()
>>> print(crs.description())
WGS 84

```

A extensão da Camada

Com o método extent() de um objeto camada podemos obter os valores máximos e mínimos das coordenadas em X (Easting ou Longitude) e Y (Northing ou Latitude). O método retorna um objeto do tipo retângulo com diversos parâmetros além de X e Y máximos e mínimos tais como area, width, height, center, invert, etc. Veja a documentação para mais informações.

```

>>> extensão=rmp.extent()
>>> min_x=extensão.xMinimum()
>>> max_x=extensão.xMaximum()
>>> min_y=extensão.yMinimum()
>>> max_y=extensão.yMaximum()
>>> print(min_x,min_y,max_x,max_y)
-70.1071697503052 -9.53901131356751 -56.7115535318741 2.213527205413

```

Quantidade de itens

O método featureCount() retorna quantos itens o objeto camada possui.

```

>>> num_elementos=rmp.featureCount()
>>> print("número de elementos: ", num_elementos)
número de elementos: 614

```

Obtendo informações dos campos de atributos

Com o método fields() obtemos a informação sobre todos os campos de atributos tais como nome, tipo etc.

```
>>> for field in rmp.fields():
...     print (field.name(),field.typeName())
CÓDIGO_OBJ int4
TOPOONIMIA text
LATITUDE float8
LONGITUDE float8
SUBST_PRIN text
SUBST_SEC text
ABREV text
STATUS_ECO text
GRAU_DE_IM text
MÉTODO_GEO text
ERRO_METOD text
DATA_CAD text
CLASSE_UTI text
TIPOLOGIA text
CLASSE_GEN text
MÓDELO_DEP text
ASSOC_GEOQ text
ROCHA_ENCA text
ROCHA_HOSP text
TEXTURA_MI text
TIPOS_ALTE text
EXTRMIN_X text
ASSOC_MINE text
ORIGEM text
UF text
MUNICIPIO text
LABEL1 text
LABEL2 text
AUTONUMBER text
```

Metadata de camada vetorial

O método htmlMetadata() gera um metadata da camada no formato html que pode ser copiado para um novo arquivo e visualizado em um navegador da web;

```
>>> metadata=rmp.htmlMetadata()
>>> print (metadata)
<html>
<body>
<h1>Informação do provedor</h1>
<hr>
<table class="list-view">
<tr><td class="highlight">Nome</td><td>Ocor. Mineral Amazonas</td></tr>
<tr><td class="highlight">Fonte</td><td>dbname='dnpm' host=amazeone.com.br
port=5432 user='droid' key='tid' checkPrimaryKeyUnicity='1' table="public"."rmp"
(geom) sql=</td></tr>
<tr><td class="highlight">Armazenamento</td><td>PostgreSQL database with PostGIS
extension</td></tr>
<tr><td class="highlight">Comentário</td><td></td></tr>
<tr><td class="highlight">Codificação</td><td>UTF-8</td></tr>
<tr><td class="highlight">Geometria</td><td>Point (Point)</td></tr>
<tr><td class="highlight">SRC</td><td>EPSG:4326 - WGS 84 - Geográfico</td></tr>
<tr><td class="highlight">Extensão</td><td>-70.1071697503052036,-9.5390113135675101
: -56.7115535318741024,2.2135272054136399</td></tr>
```

```

<tr><td class="highlight">Unidade</td><td>graus</td></tr>
<tr><td class="highlight">Contagem de feições</td><td>614</td></tr>
</table>
<br><br><h1>Identificação</h1>
<hr>
<table class="list-view">
<tr><td class="highlight">Identifier</td><td></td></tr>
<tr><td class="highlight">Parent Identifier</td><td></td></tr>
<tr><td class="highlight">Title</td><td></td></tr>
<tr><td class="highlight">Type</td><td>dataset</td></tr>
<tr><td class="highlight">Language</td><td></td></tr>
<tr><td class="highlight">Abstract</td><td></td></tr>
<tr><td class="highlight">Categories</td><td></td></tr>
<tr><td class="highlight">Keywords</td><td></td></tr>
</td></tr>
</table>
<br><br>
<h1>Extensão</h1>
<hr>
<table class="list-view">
<tr><td class="highlight">CRS</td><td>EPSG:4326 - WGS 84 - Geographic</td></tr>
<tr><td class="highlight">Spatial Extent</td><td></td></tr>
<tr><td class="highlight">Temporal Extent</td><td></td></tr>
</table>
<br><br>
<h1>Acesso</h1>
<hr>
<table class="list-view">
<tr><td class="highlight">Fees</td><td></td></tr>
<tr><td class="highlight">Licenses</td><td></td></tr>
<tr><td class="highlight">Rights</td><td></td></tr>
<tr><td class="highlight">Constraints</td><td></td></tr>
</table>
<br><br>
<h1>Campos</h1>
<hr>
<table class="list-view">
<tr><td class="highlight">Contagem</td><td>29</td></tr>
</table>
<br><table width="100%" class="tabular-view">
<tr><th>Campo</th><th>Tipo</th><th>Comprimento</th><th>Precisão</th><th>Comentário</th></tr>
<tr ><td>CODIGO_OBJ</td><td>int4</td><td>-1</td><td>0</td><td></td></tr>
<tr class="odd-row"><td>TOPONIMIA</td><td>text</td><td>-1</td><td>-1</td><td></td></tr>
<tr ><td>LATITUDE</td><td>float8</td><td>-1</td><td>-1</td><td></td></tr>
<tr class="odd-row"><td>LONGITUDE</td><td>float8</td><td>-1</td><td>-1</td><td></td></tr>
<tr ><td>SUBST_PRIN</td><td>text</td><td>-1</td><td>-1</td><td></td></tr>
<tr class="odd-row"><td>SUBST_SEC</td><td>text</td><td>-1</td><td>-1</td><td></td></tr>
<tr ><td>ABREV</td><td>text</td><td>-1</td><td>-1</td><td></td></tr>
<tr class="odd-row"><td>STATUS_ECO</td><td>text</td><td>-1</td><td>-1</td><td></td></tr>
<tr ><td>GRAU_DE_IM</td><td>text</td><td>-1</td><td>-1</td><td></td></tr>
<tr class="odd-row"><td>METODO_GEO</td><td>text</td><td>-1</td><td>-1</td><td></td></tr>
<tr><td>...</td></tr>

```

```

<tr class="odd-row"><td>LABEL2</td><td>text</td><td>-1</td><td>-1</td><td></td></tr>
<tr ><td>AUTONUMBER</td><td>text</td><td>-1</td><td>-1</td><td></td></tr>
</table>
<br><br><h1>Contatos</h1>
<hr>
<p>No contact yet.</p><br><br>
<h1>Links</h1>
<hr>
<p>No links yet.</p>
<br><br>
<h1>Histórico</h1>
<hr>
<p>No history yet.</p>
<br><br>

</body>
</html>

```

Essa informação acima apareceria no navegador assim:

Informação do provedor

Nome	Ocor. Mineral Amazonas
Fonte	dbname='dnpm' host=amazeone.com.br port=5432 user='droid' key='tid' checkPrimaryKeyUnicity='1' table="public"."rmp" (geom) sql=
Armazenamento	PostgreSQL database with PostGIS extension
Comentário	
Codificação	UTF-8
Geometria	Point (Point)
SRC	EPSG:4326 - WGS 84 - Geográfico
Extensão	-70.1071697503052036,-9.5390113135675101 : - 56.7115535318741024,2.2135272054136399
Unidade	graus
Contagem de feições	614

Identificação

Identifier	
Parent Identifier	

Title

Type dataset

Language

Abstract

Categories

Keywords

Extensão

CRS EPSG:4326 - WGS 84 - Geographic

Spatial Extent

Temporal Extent

Acesso

Fees

Licenses

Rights

Constraints

Campos

Contagem

29

Campo	Tipo	Comprimento	Precisão	Comentário
CODIGO_OBJ	int4	-1	0	
TOPONIMIA	text	-1	-1	
LATITUDE	float8	-1	-1	
LONGITUDE	float8	-1	-1	
SUBST_PRIN	text	-1	-1	
SUBST_SEC	text	-1	-1	
ABREV	text	-1	-1	
STATUS_ECO	text	-1	-1	
GRAU_DE_IM	text	-1	-1	
METODO_GEO	text	-1	-1	
ERRO_METOD	text	-1	-1	
DATA_CAD	text	-1	-1	
CLASSE_UTI	text	-1	-1	
TIPOLOGIA	text	-1	-1	
CLASSE_GEN	text	-1	-1	
MODELO_DEP	text	-1	-1	
ASSOC_GEOQ	text	-1	-1	
ROCHA_ENCA	text	-1	-1	
ROCHA_HOSP	text	-1	-1	
TEXTURA_MI	text	-1	-1	
TIPOS_ALTE	text	-1	-1	
EXTRMIN_X_	text	-1	-1	
ASSOC_MINE	text	-1	-1	
ORIGEM	text	-1	-1	
UF	text	-1	-1	
MUNICIPIO	text	-1	-1	
LABEL1	text	-1	-1	
LABEL2	text	-1	-1	
AUTONUMBER	text	-1	-1	

Contatos

No contact yet.

Links

No links yet.

Histórico

No history yet.

Obtendo os elementos de cada item da camada vetorial

Com o método `getFeatures()` carregamos todos os dados da camada, onde cada item é armazenado como uma lista. O código abaixo imprime cada um dos itens em formato de lista.

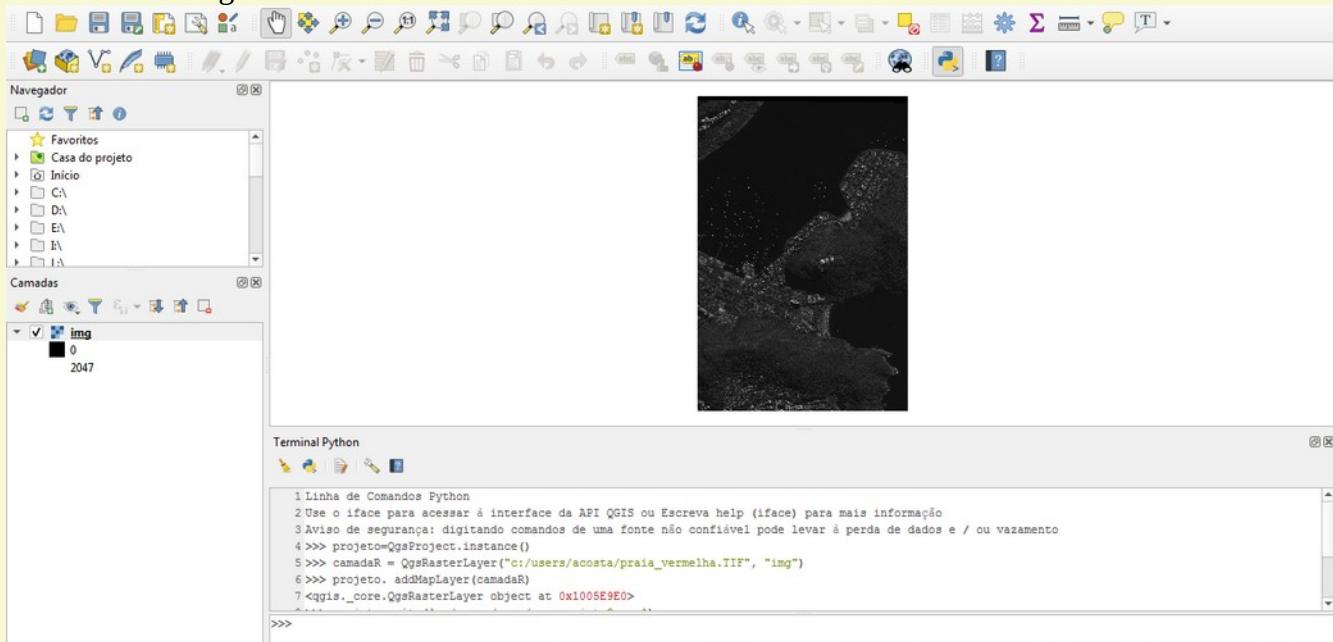
```
>>> elementos=rmp.getFeatures()
>>> for e in elementos:
...     attr=e.attributes()
...     print (attr)
[0, 'AREAL DO FOLE I', -2.70984, -59.67626, 'Areia', NULL, 'ar', NULL, NULL, NULL,
NULL, NULL, 'Material de uso na construção civil', NULL, NULL, NULL, NULL, NULL,
NULL, NULL, NULL, NULL, NULL, NULL, NULL, '1 ar', '1 ar      AREAL DO FOLE
I', '1']
[0, 'FAZENDA DAS LARANJEIRAS', -2.71679265, -59.66741255, 'Areia', NULL, 'ar', 'Não
explotado', 'Ocorrência', 'GPS', NULL, NULL, 'Material de uso na construção civil',
NULL, '2
ar', '2 ar      FAZENDA DAS LARANJEIRAS', '2']
...
[46429, 'SERRA DO MEIO', 1.84472222, -68.20694444, 'Au', NULL, 'Au', 'Garimpo',
'Depósito', 'Levantamento em Carta 1:100.000', '100 a 400 m', NULL, 'Metais
nobres', NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, 'RN_TA', 'AM',
'SAO GABRIEL DA CACHOEIRA', '614 Au', '614 Au      SERRA DO MEIO', '614']
```

3.3 Interagindo com informações de objetos da classe Raster

De forma semelhante ao que foi feito acima, podemos extrair informações relevantes de um objeto raster também tais como dimensões, resoluções, número de bandas, valor de um pixel, etc. Vamos carregar uma imagem inicialmente.

```
>>> projeto=QgsProject.instance()
>>> camadaR = QgsRasterLayer("c:/users/voce/praiavermelha.TIF", "img")
>>> projeto.addMapLayer(camadaR)
```

Isso abrirá a imagem como mostrado abaixo:



Informações de dimensão do objeto Raster

Podemos acessar informações de parâmetros dimensionais de uma imagem raster usando métodos específicos para o tal.

```
>>> camadaR.width(), camadaR.height() #largura e altura  
(5439, 8192)  
>>> camadaR.extent() #extensão da imagem na unidade da coordenada  
<QgsRectangle: 687173.999999998358468 7459532.40000000037252903,  
688805.6999999983701855 7461990>  
>>> camadaR.crs().description() #sistema de referência  
'WGS 84 / UTM zone 23S'  
>>> camadaR.rasterUnitsPerPixelX() #resolução em X  
0.29999999999999144  
>>> camadaR.rasterUnitsPerPixelY() #resolução em Y  
0.2999999999999545
```

Essas importantes informações sobre o raster poderão ser usadas para análises espaciais futuras. Existem outras formas de usar os métodos para obtermos a mesma informação. Podemos calcular a resolução em X usando:

```
>>> (camadaR.extent().xMaximum()-camadaR.extent().xMinimum())/camadaR.width()  
0.2999999999999144
```

em vez de usar o método rasterUnitsPerPixelX(). Podemos também visualizar informações usando o método htmlMetadata().

```
>>> camadaR.htmlMetadata()
```

Informação do provedor

Nome	img
Caminho	/home/voce/curso/praiavermelha.TIF
SRC	EPSG:32723 - WGS 84 / UTM zone 23S - Projetado
Extensão	687173.99999999835847,7459532.4000000003725290 : 688805.699999998370185,7461990.0000000000000000
Unidade	metros
Largura	5439
Altura	8192
tipo de dado	UInt16 - Inteiro de 16 bits sem sinal
Descrição do driver GDAL	GTiff
Metadados do driver GDAL	GeoTIFF
Descrição do registro	/home/andre/curso/praiavermelha.TIF
Compressão	<ul style="list-style-type: none">• STATISTICS_APPROXIMATE=YES• STATISTICS_MAXIMUM=2047• STATISTICS_MEAN=220.86768822394• STATISTICS_MINIMUM=0• STATISTICS_STDDEV=156.88305888003• STATISTICS_VALID_PERCENT=100
Banda 1	<ul style="list-style-type: none">• AREA_OR_POINT=Area
Mais informação	

- TIFFTAG_COPYRIGHT=(C) COPYRIGHT 2016 DigitalGlobe, Inc., Longmont CO USA 80503
- TIFFTAG_DATETIME=2016:09:02 04:03:04
- TIFFTAG_IMAGEDESCRIPTION={ bandList = [1;]}
- TIFFTAG_MAXSAMPLEVALUE=2047
- TIFFTAG_MINSAMPLEVALUE=0

Dimensões X: 5439 Y: 8192 Bandas: 1

Origem 687174,7.46199e+06

Tamanho do Pixel 0.29999999999999889,-0.29999999999999889

Identificação

Identifier

Parent Identifier

Title

Type

Language

Abstract

Categories

Keywords

Extensão

CRS

Spatial Extent

Temporal Extent

Acesso

Fees

Licenses

Rights

Constraints

Bandas

Contagem de bandas

1

Número

Banda

Sem Dados

Mín

Máx

1

Banda 1

n/a

n/a

n/a

Contatos

No contact yet.

Referências

No links yet.

Histórico

No history yet.

Vamos ver agora métodos para raster de uma e de mais de uma banda.

Raster com uma banda de valores

Os métodos abaixo informam o número de bandas e o tipo da imagem raster. 0 para cinza ou não definido de banda única, 1 para paletado de banda única e 2 para multibanda.

```
>>> camadaR.bandCount()
1
>>> camadaR.rasterType()
0
```

A função `dataProvider()` funciona como uma interface entre o objeto raster os seus dados individuais, seu método `sample()` toma dois valores, um objeto ponto (coordenadas XZ) e o número da banda. Se a coordenada for dentro da imagem e a banda existir o resultado será um tuple com o valor do pixel e se o dado é verdadeiro ou não.

```
>>> valor = camadaR.dataProvider().sample(QgsPointXY(687567, 7460876), 1)
>>> valor
(163.0, True)
>>> valor2 = camadaR.dataProvider().sample(QgsPointXY(687567, 7463876), 1)
>>> valor2
(nan, False)
```

A rampa de cor assinalada ao objeto raster pode ser checada usando o método `type()` do método `renderer()`. O tipo `singlebandgray` é o padrão inicial.

```
>>> camadaR.renderer().type()
'singlebandgray'
```

Podemos alterar via python a rampa de cores, o processo é mostrado abaixo. O processo envolve na criação de um objetos do tipo `ColorRampShader` e definimos a rampa de cor de preenchimento como sendo do tipo `interpolado`.

```
>>> fcn = QgsColorRampShader()
>>> fcn.setColorRampType(QgsColorRampShader.Interpolated)
```

Criamos agora uma lista com as cores representando os dois valores extremos do raster (0 e 2046 que serão interpolados entre azul e amarelo. Em seguida adicionamos esta lista como item do `ColorRampShader` criado acima.

```
>>> lista = [QgsColorRampShader.ColorRampItem(0, QColor(0,0,255)),
QgsColorRampShader.ColorRampItem(2046, QColor(255,255,0))]
>>> fcn.setColorRampItemList(lista)
```

O próximo passo é criarmos o `RasterShader` (preenchedor de cor) e associarmos o `RampShader` a ele.

```
>>> shader = QgsRasterShader()
>>> shader.setRasterShaderFunction(fcn)
```

Finalmente criamos o objeto renderizador de cor com: dados do objeto raster, banda 1 e shader acima.

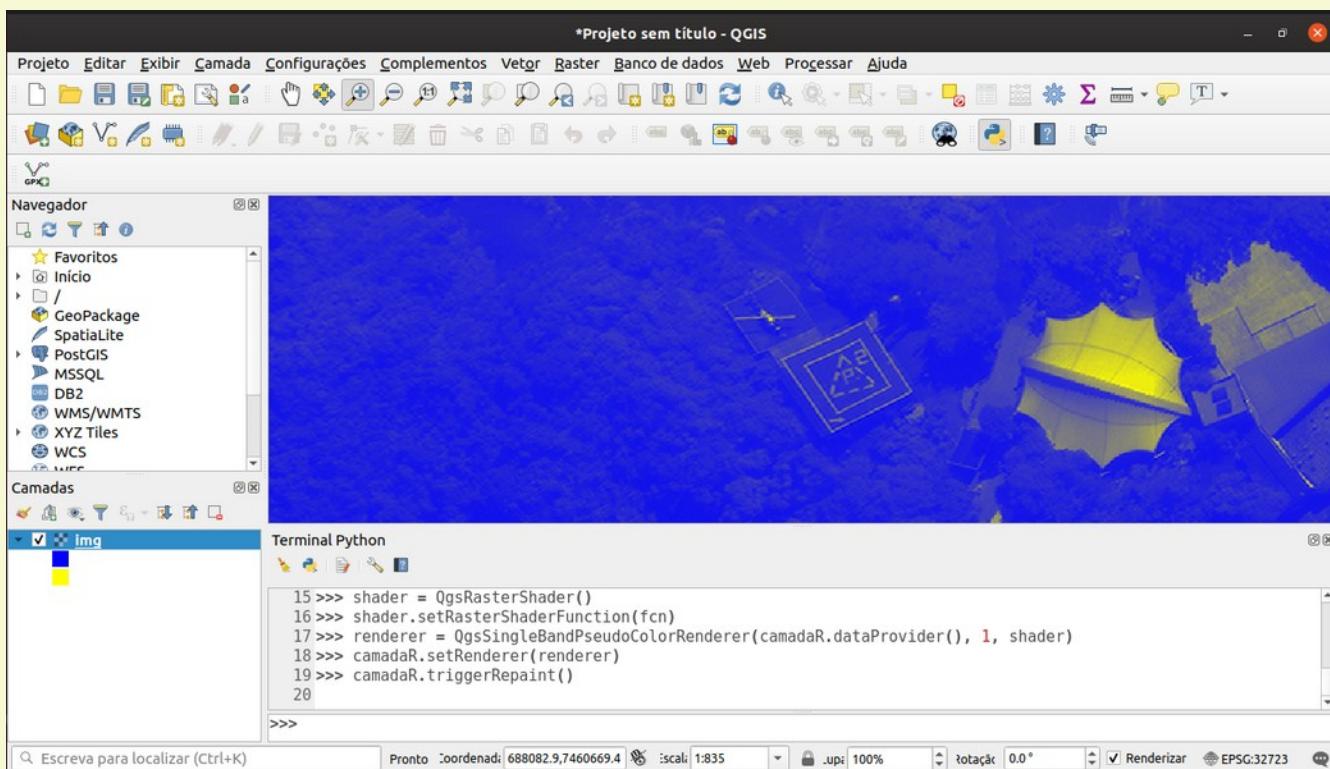
Em seguida aplicamos este ao objeto raster e chamamos a repintura do objeto

```
>>> renderer = QgsSingleBandPseudoColorRenderer(camadaR.dataProvider(), 1, shader)
>>> camadaR.setRenderer(renderer)
>>> camadaR.triggerRepaint()
```

Se chamarmos o tipo novamente podemos ver a mudança.

```
>>> camadaR.renderer().type()
'singlebandpseudocolor'
```

O resultado final é mostrado num detalhe do raster (zoom in) na imagem abaixo.



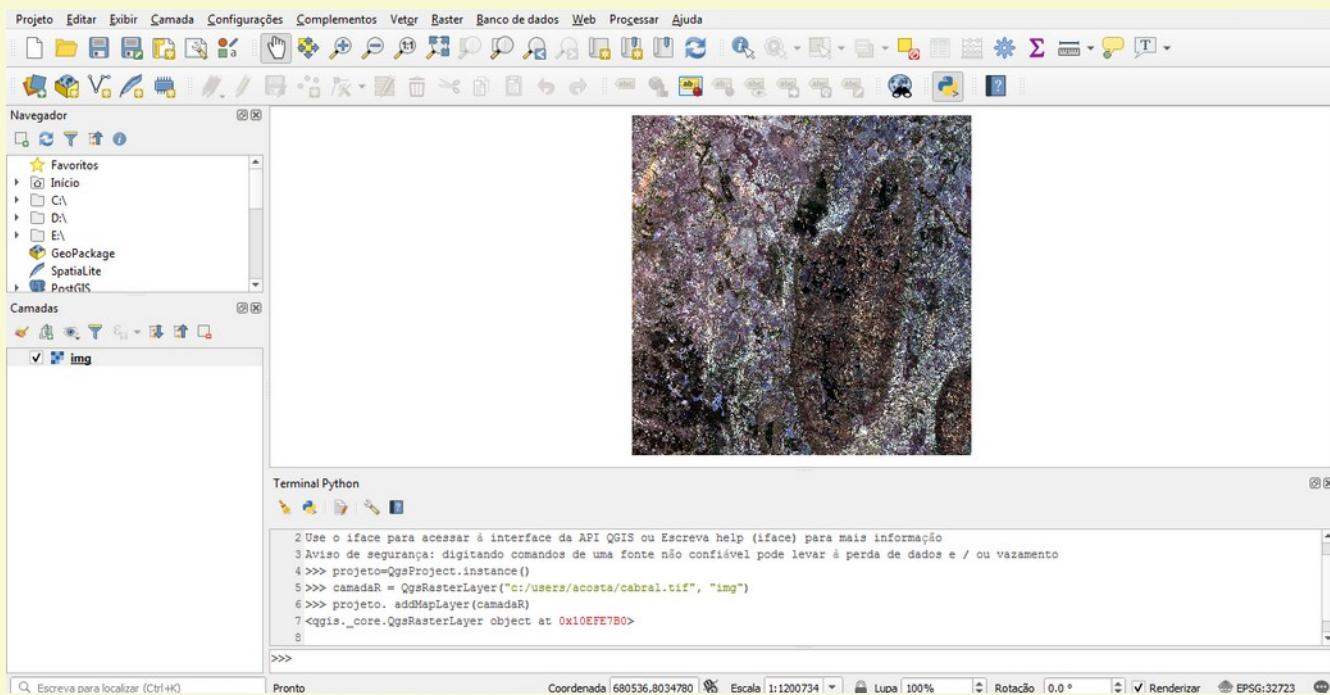
Raster com mais de uma banda de valores

Vamos trabalhar um pouco agora com imagem raster de 3 bandas. Carregamos o raster de forma similar e vamos extrair algumas de suas informações.

```

>>> projeto=QgsProject.instance()
>>> camadaR = QgsRasterLayer("c:/users/você/cabral.tif", "img")
>>> projeto.addMapLayer(camadaR)

```

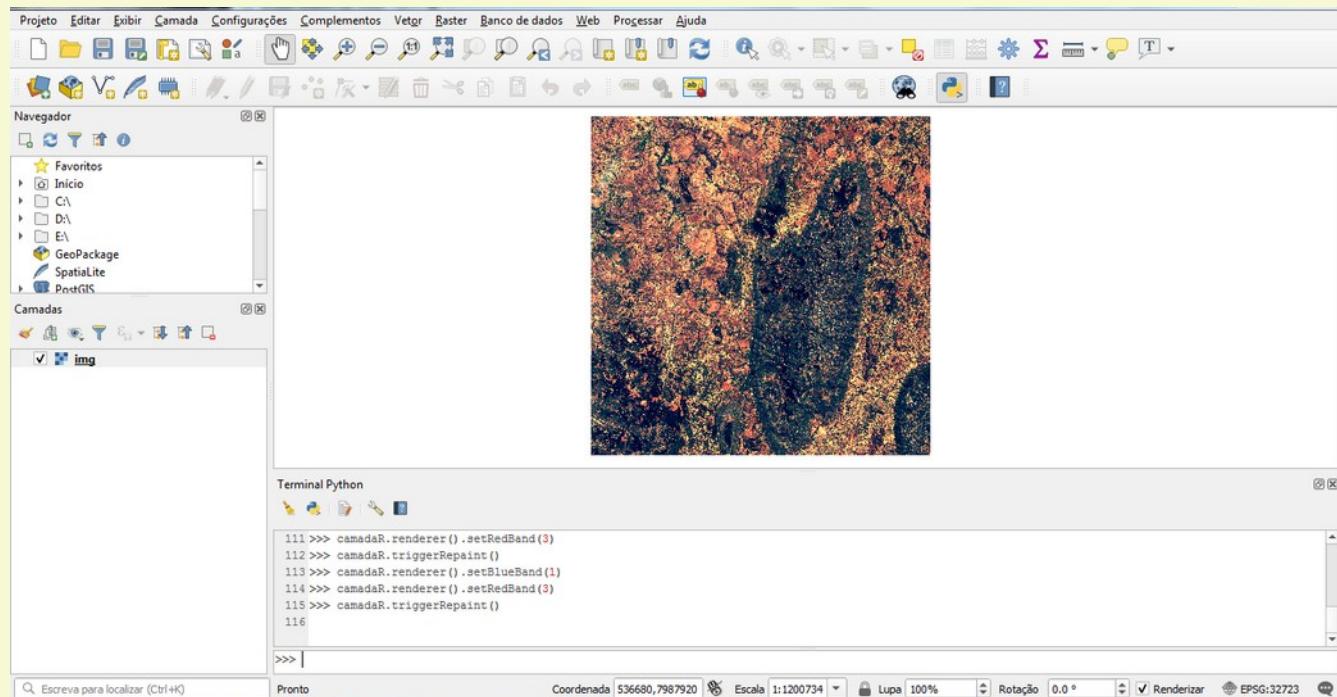


Podemos ver algumas das informações usando:

```
>>> camadaR.bandCount() # número de bandas
3
>>> camadaR.rasterType() # 2 para multi banda
2
>>> valor = camadaR.dataProvider().sample(QgsPointXY(554729, 8044946), 1) # valor
do pixel na banda 1
>>> valor
(791.0, True)
>>> valor = camadaR.dataProvider().sample(QgsPointXY(554729, 8044946), 2) # valor
do pixel na banda 2
>>> valor
(668.0, True)
>>> valor = camadaR.dataProvider().sample(QgsPointXY(554729, 8044946), 3) # valor
do pixel na banda 3
>>> valor
(535.0, True)
>>> camadaR.renderer().type()
'multibandcolor'
```

Vamos ver abaixo como modificar a imagem para que a banda 1 fique no canal azul (B) e a banda 3 fique no canal Vermelho (R).

```
>>> camadaR.renderer().setBlueBand(1)
>>> camadaR.renderer().setRedBand(3)
>>> camadaR.triggerRepaint()
```



Note que o histograma da imagem não foi apropriadamente ajustado porque ainda usa os valores de máximo e mínimo das bandas anteriores.

3.4 Criando objeto vetorial

Vamos agora ver os passos para criarmos objetos vetoriais usando python no Qgis. Vamos criar objetos do tipo ponto, linha e polígono para ilustrar o processo.

Ponto

Primeiro definimos o objeto ponto com CRS 4326 (WGS84) com o nome Cidades na memória. Nesse objeto usamos um dataProvider para criar os campos de atributos do objeto vetorial ponto com três atributos (nome, população e IDH) e adicionamos eles no objeto ponto (vponto).

```
>>> vponto = QgsVectorLayer("Point?crs=EPSG:4326", "Cidades", "memory")
>>> dPr = vponto.dataProvider()
>>> dPr.addAttribute([QgsField("nome", QVariant.String), QgsField("populacao",
QVariant.Int), QgsField("idh", QVariant.Double)])
>>> vponto.updateFields()
```

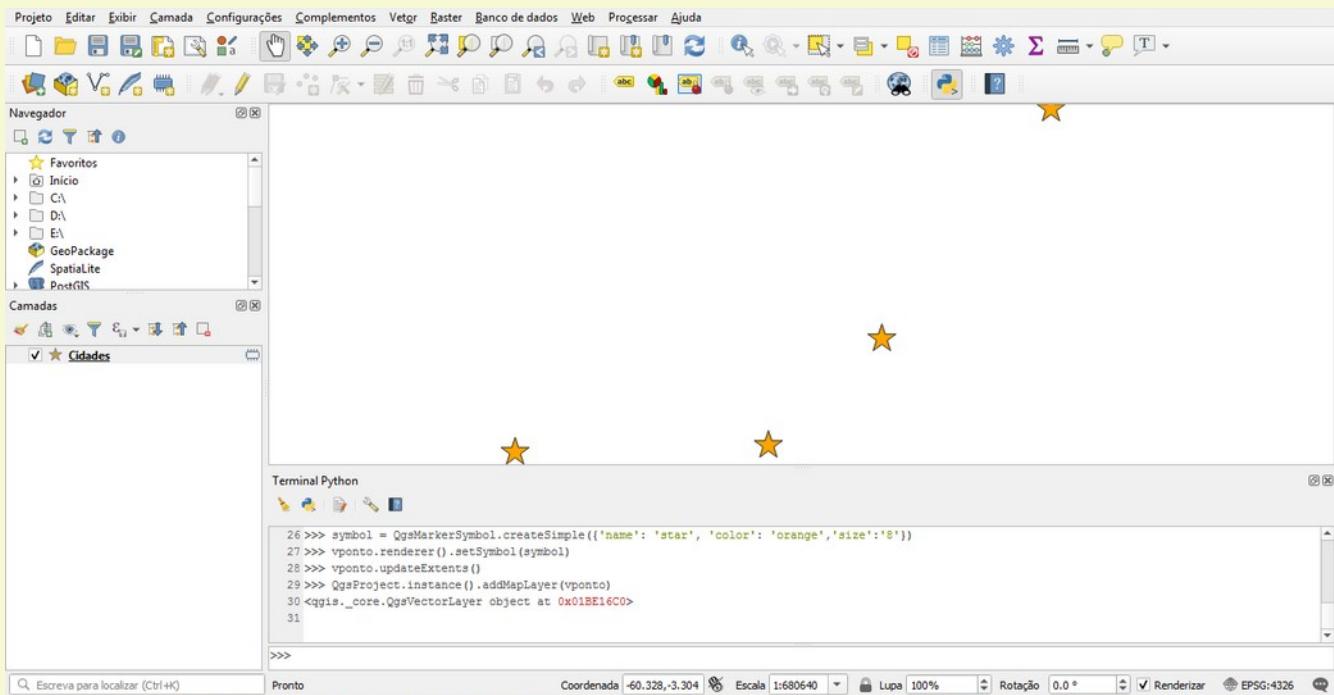
Uma vez criado o objeto ponto e seus campos de atributo vamos adicionar dados nele usando um objeto feature (elemento). Definimos a geometria que será um ponto nesse caso com coordenadas x e y e também adicionaremos os atributos deste ponto.

```
>>> elem = QgsFeature()
>>> elem.setGeometry(QgsGeometry.fromPointXY(QgsPointXY(-59.9936,-3.0925)))
>>> elem.setAttributes(["Manaus", 2182763, 0.737])
>>> dPr.addFeature(elem)
>>> elem.setGeometry(QgsGeometry.fromPointXY(QgsPointXY(-60.6253,-3.2872)))
>>> elem.setAttributes(["Manacapuru ", 97377, 0.614])
>>> dPr.addFeature(elem)
>>> elem.setGeometry(QgsGeometry.fromPointXY(QgsPointXY(-60.1883,-3.2756)))
>>> elem.setAttributes(["Iranduba ", 48296, 0.613])
>>> dPr.addFeature(elem)
>>> elem.setGeometry(QgsGeometry.fromPointXY(QgsPointXY(-59.7014,-2.6968)))
>>> elem.setAttributes(["Rio Preto Da Eva ", 33347, 0.611])
>>> dPr.addFeature(elem)
```

Para finalizar vamos configurar a aparência do símbolo mostrado no mapa como estrelas de laranjas e de tamanho 8. Atualizamos a extensão do mapa e adicionamos nosso objeto no mapa.

```
>>> symbol = QgsMarkerSymbol.createSimple({'name': 'star', 'color':
'orange','size':'8'})
>>> vponto.renderer().setSymbol(symbol)
>>> vponto.updateExtents()
>>> QgsProject.instance().addMapLayer(vponto)
```

A aparência do mapa será conforme a imagem mostrada abaixo.



Os parâmetros abaixo podem ser usados com `QgsMarkerSymbol.createSimple()`:

```

'angle': '0',
'color': '255,165,0,255',
'horizontal_anchor_point': '1',
'joinstyle': 'bevel',
'name': 'star',
'offset': '0,0',
'offset_map_unit_scale': '3x:0,0,0,0,0,0',
'offset_unit': 'MM',
'outline_color': '35,35,35,255',
'outline_style': 'solid',
'outline_width': '0',
'outline_width_map_unit_scale': '3x:0,0,0,0,0,0',
'outline_width_unit': 'MM',
'scale_method': 'diameter',
'size': '8',
'size_map_unit_scale': '3x:0,0,0,0,0,0',
'size_unit': 'MM',
'vertical_anchor_point': '1'

```

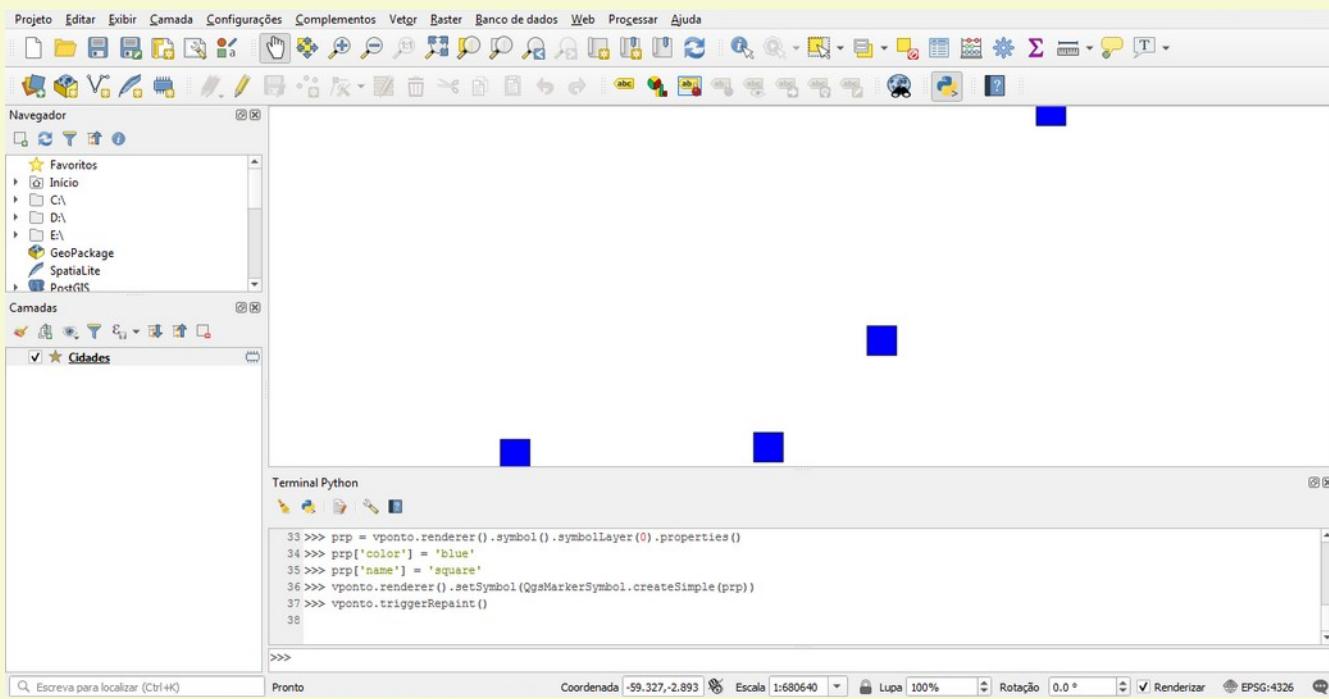
Veja a documentação para cada uma das opções que podem ser usadas com cada parâmetro listado acima. Vamos Modificar a aparência do símbolo que criamos acima.

```

>>> prp = vponto.renderer().symbol().symbolLayer(0).properties()
>>> prp['color'] = 'blue'
>>> prp['name'] = 'square'
>>> vponto.renderer().setSymbol(QgsMarkerSymbol.createSimple(prp))
>>> vponto.triggerRepaint()

```

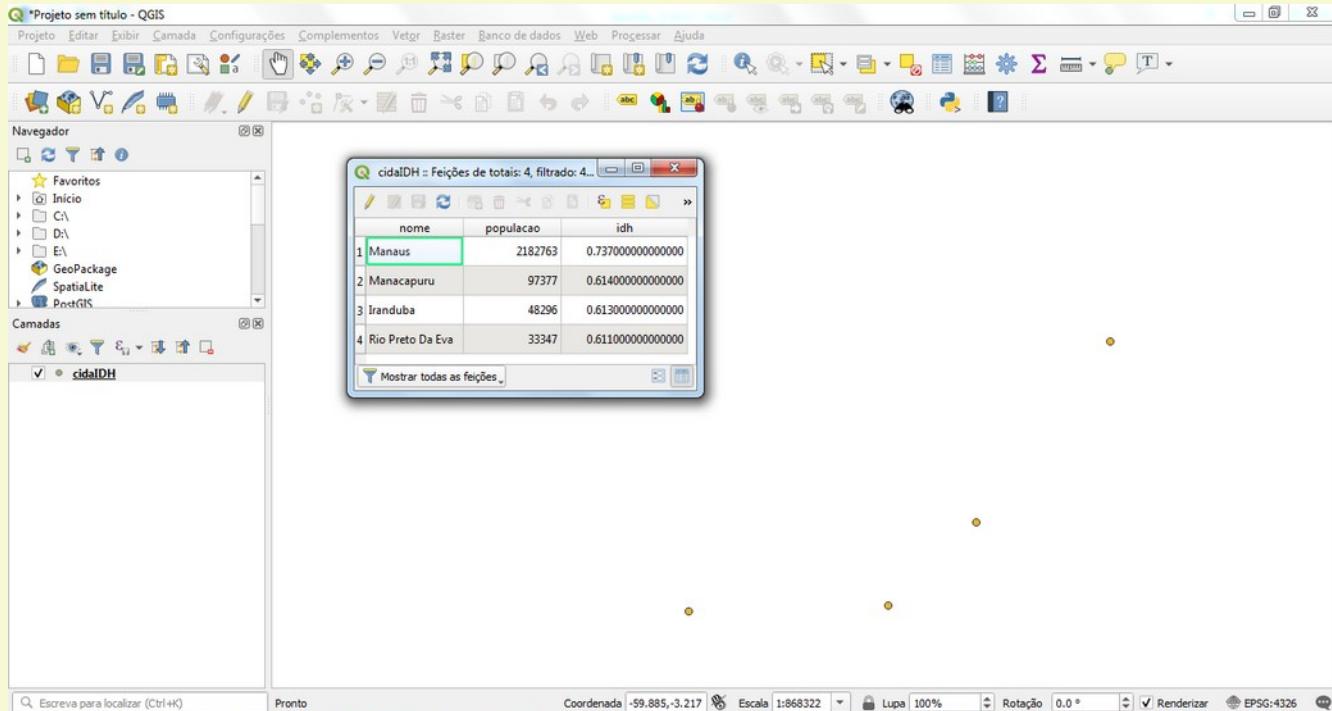
Veja o resultado abaixo.



Finalizamos mostrando como escrever o nosso objeto ponto em um arquivo em um arquivo.

```
>>> QgsVectorFileWriter.writeAsVectorFormat(vponto, 'c:/users/voce/cidaIDH.shp',
'utf-8', driverName='ESRI Shapefile')
```

Ao abrirmos o arquivo recém criado e visualizar sua tabela de atributos teremos algo similar ao mostrado abaixo.



Linha

De forma semelhante ao que fizemos com pontos, criar linhas usando python/Qgis é só uma questão de usarmos uma série (lista) de pontos para cada elemento criado.

Definimos o objeto linha (linestring) com CRS 4326 (WGS84) com o nome Vias na memória. Criamos o dataProvider para adicionar os campos de atributos do objeto vetorial linestring com dois atributos (nome e número) e adicionamos eles no objeto linestring (vlinha).

```
>>> vlinha = QgsVectorLayer("LineString?crs=EPSG:4326", "Vias", "memory")
>>> dPr = vlinha.dataProvider()
>>> dPr.addAttribute([QgsField("nome", QVariant.String), QgsField("número",
QVariant.Int)])
>>> vlinha.updateFields()
```

Adicionamos as linhas usando um objeto feature (elemento). Mas antes criamos a lista de pontos que farão parte de cada um dos elemento do tipo linha e também inserimos os atributos de cada elemento. Vamos inserir três linhas.

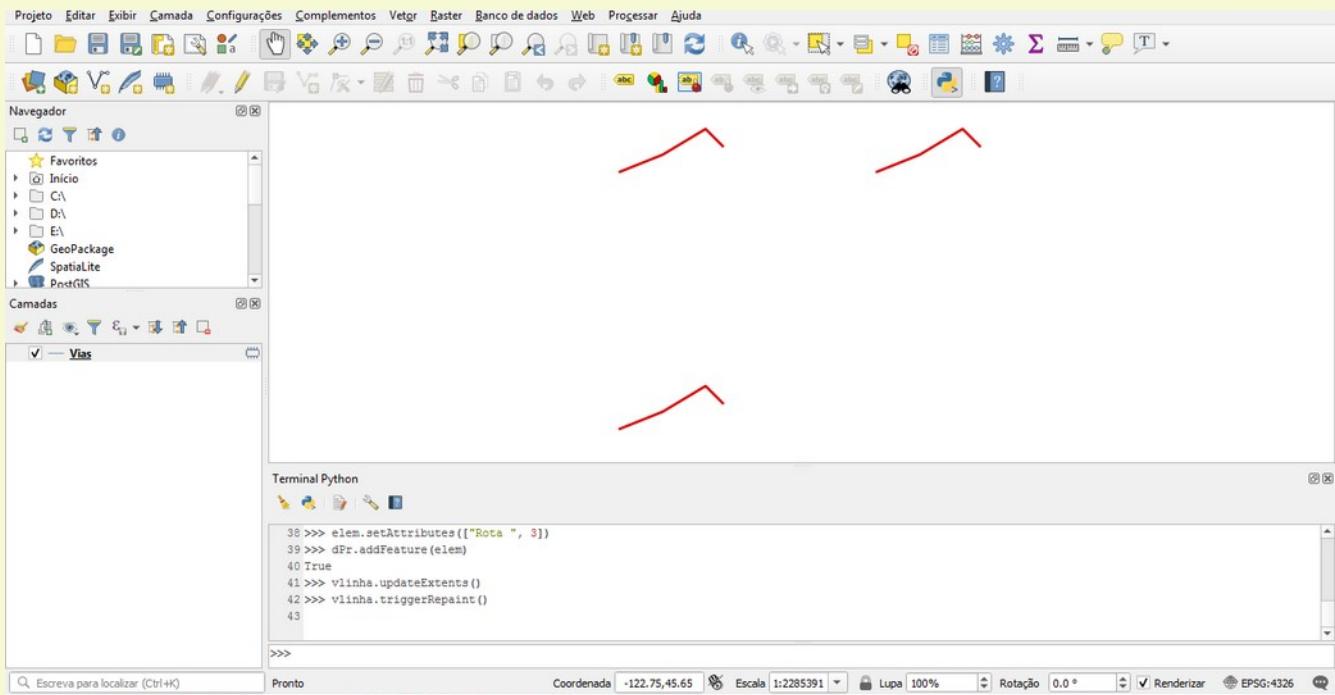
```
>>> elem = QgsFeature()
>>> pontos =[QgsPoint(-124,48.4), QgsPoint(-123.5,48.6 ), QgsPoint(-
123,48.9),QgsPoint(-122.8,48.7)]
>>> elem.setGeometry(QgsGeometry.fromPolyline(pontos))
>>> elem.setAttributes(["Rota ", 1])
>>> dPr.addFeature(elem)
>>> pontos =[QgsPoint(-121,48.4), QgsPoint(-120.5,48.6 ), QgsPoint(-
120,48.9),QgsPoint(-119.8,48.7)]
>>> elem.setGeometry(QgsGeometry.fromPolyline(pontos))
>>> elem.setAttributes(["Rota ", 2])
>>> dPr.addFeature(elem)
>>> pontos =[QgsPoint(-124,45.4), QgsPoint(-123.5,45.6 ), QgsPoint(-
123,45.9),QgsPoint(-122.8,45.7)]
>>> elem.setGeometry(QgsGeometry.fromPolyline(pontos))
>>> elem.setAttributes(["Rota ", 3])
>>> dPr.addFeature(elem)
```

Atualizamos a extensão do objeto e o adicionamos ao mapa (canvas).

```
>>> vlinha.updateExtents()
>>> QgsProject.instance().addMapLayer(vlinha)
```

Podemos mudar a aparência de nosso objeto vetorial linestring usando:

```
>>> vlinha.renderer().symbol().setWidth(0.7)
>>> vlinha.renderer().symbol().setColor(QColor(224,0,0))
>>> vlinha.triggerRepaint()
```



Para finalizar, vamos gravar o recém criado vetor num arquivo do tipo shapefile.

```
>>> QgsVectorFileWriter.writeAsVectorFormat(vlinha, 'c:/users/voce/vias.shp', 'utf-8', driverName='ESRI Shapefile')
```

Polígono

Criamos polígonos usando python/Qgis de forma idêntica à forma que criamos linhas só que nesse caso o último ponto se liga ao primeiro ponto informado.

Definimos o objeto polígono com CRS 4326 (WGS84) com o nome Fazendas na memória. Criamos o dataProvider para adicionar os campos de atributos do objeto vetorial polígono com dois atributos (nome e número) e adicionamos eles no objeto polígono (vpgon).

```
>>> vpgon = QgsVectorLayer("Polygon?crs=EPSG:4326", "Fazendas", "memory")
>>> dPr = vpgon.dataProvider()
>>> dPr.addAttribute([QgsField("nome", QVariant.String), QgsField("número", QVariant.Int)])
>>> vpgon.updateFields()
```

Adicionamos os polígonos usando um objeto feature (elemento). Mas antes criamos a lista de pontos que farão parte de cada um dos elementos do tipo polígono (em colchete duplo) e também inserimos os atributos de cada elemento. Vamos inserir dois polígonos.

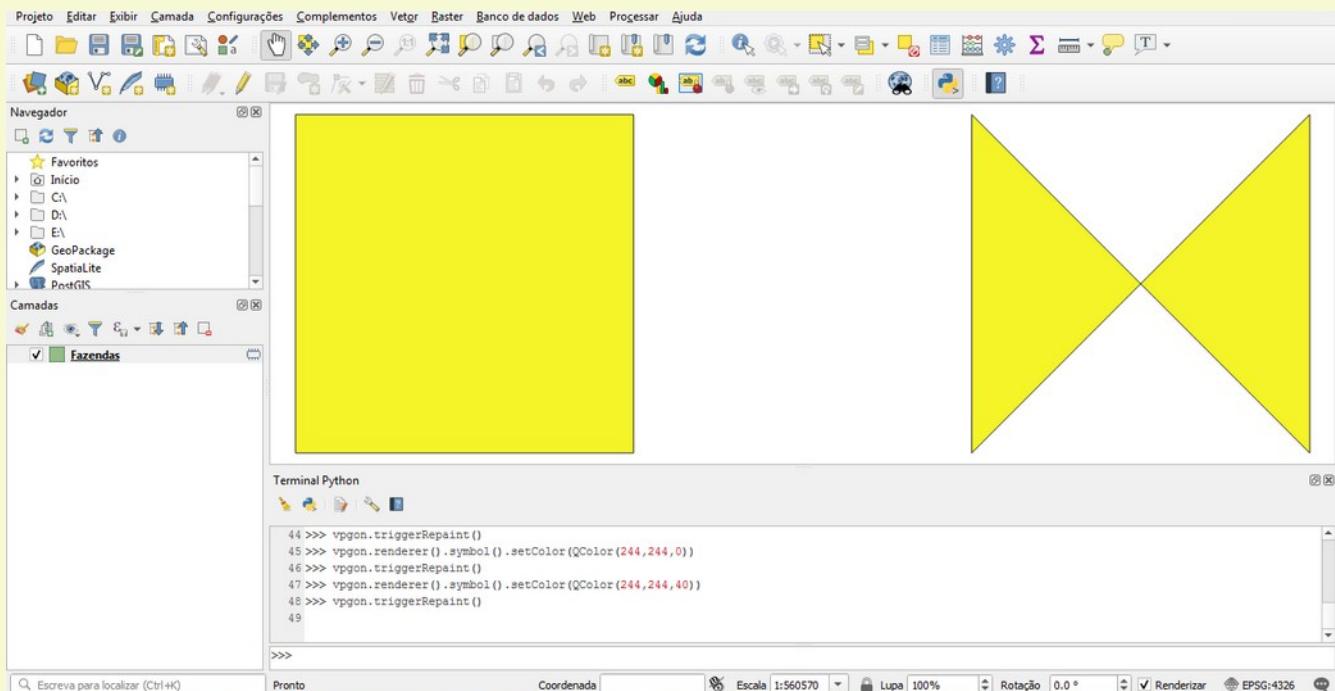
```
>>> elem = QgsFeature()
>>> pontos = [[QgsPointXY(-124,48), QgsPointXY(-123,48), QgsPointXY(-123,49), QgsPointXY(-124,49)]]
>>> elem.setGeometry(QgsGeometry.fromPolygonXY(pontos))
>>> elem.setAttributes(["Fazenda Abre Campo ", 1])
>>> dPr.addFeature(elem)
>>> pontos = [[QgsPointXY(-122,48), QgsPointXY(-121,49), QgsPointXY(-121,48), QgsPointXY(-122,49)]]
>>> elem.setGeometry(QgsGeometry.fromPolygonXY(pontos))
>>> elem.setAttributes(["Fazenda Vista Alegre ", 2])
>>> dPr.addFeature(elem)
```

Atualizamos a extensão do objeto e o adicionamos ao mapa (canvas).

```
>>> vpgon.updateExtents()  
>>> QgsProject.instance().addMapLayer(vpgon)
```

Podemos mudar a aparência de nosso objeto vetorial polígono usando:

```
>>> vpgon.renderer().symbol().setColor(QColor(224,224,40))  
>>> vpgon.triggerRepaint()
```



Gravamos o vetor recém criado num arquivo do tipo shapefile usando.

```
>>> QgsVectorFileWriter.writeAsVectorFormat(vpgon, 'c:/users/voce/fazendas.shp',  
'utf-8', driverName='ESRI Shapefile')
```

3.5 Criando objeto raster

Imagens raster também podem ser criadas via script de forma bem eficiente usando uma lista de dados pontuais com um determinado valor. Vamos aqui criar um raster mostrando a temperatura média de uma área com base em informações pontuais de vários locais. O arquivo CSV **tfinal.csv** tem os dados com coordenadas, e respectivos valores. Vamos carregar a informação em um objeto do tipo **QgsInterpolator** camada de dados (layerData).

```
>>> uri="file:///c:/users/voce/tfinal.csv?  
type=csv&xField=LONGITUDE&yField=LATITUDE&crs=epsg:4326"  
>>> camada = QgsVectorLayer(uri, 'Converte', "delimitedtext")  
>>> c_data = QgsInterpolator.LayerData()  
>>> c_data.source = camada  
>>> c_data.zCoordInterpolation = False  
>>> c_data.interpolationAttribute = 6  
>>> c_data.sourceType = QgsInterpolator.SourcePoints
```

Executaremos a interpolação usando o inverso da distância ponderada (IDW) ao quadrado (coeficiente 2).

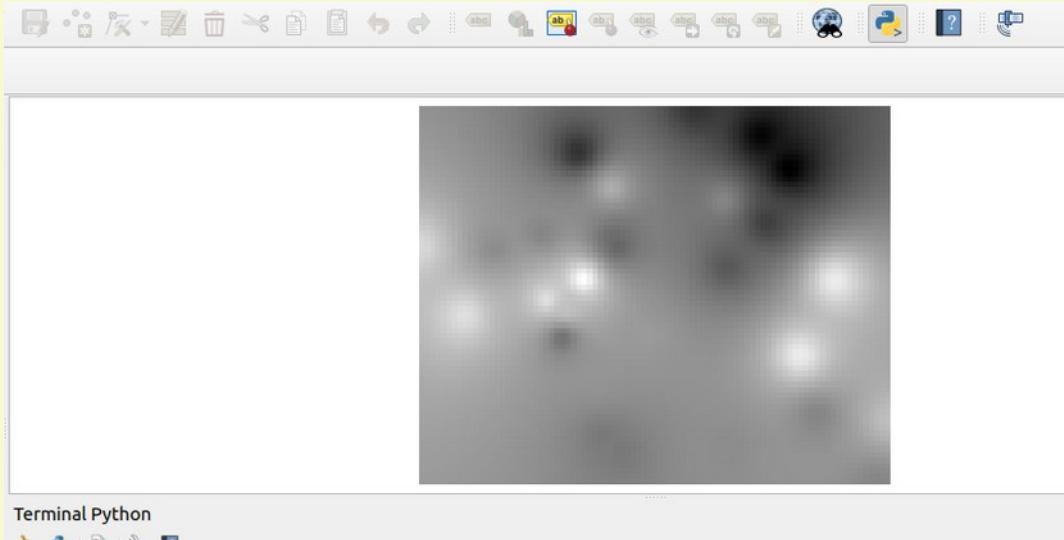
```
>>> interpolado = QgsIDWInterpolator([c_data])
>>> interpolado.setDistanceCoefficient(2)
```

Agora definimos qual arquivo será usado e os parâmetros do grid a ser usado.

```
>>> arquivo = "c:/users/voce/rasterDeTeste.asc"
>>> rect = camada.extent()
>>> res = 0.01
>>> ncol = int( ( rect.xMaximum() - rect.xMinimum() ) / res )
>>> nrows = int( (rect.yMaximum() - rect.yMinimum() ) / res)
>>> saida = QgsGridFileWriter(interpolado,arquivo,rect,ncol,nrows)
>>> saida.writeFile()
```

Carregamos o arquivo do grid usando.

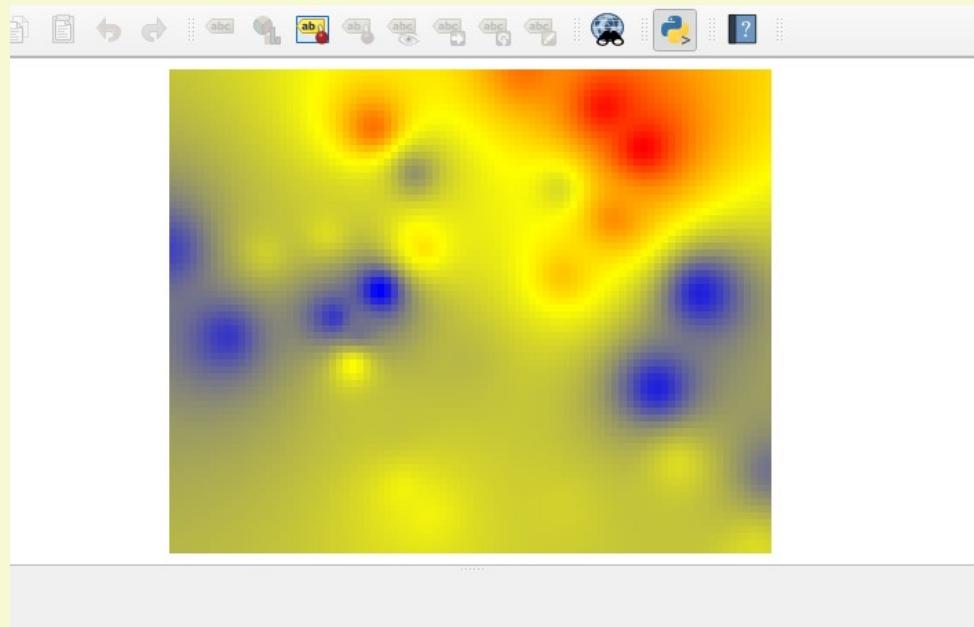
```
>>> camadaR = iface.addRasterLayer(arquivo, "raster_interpolado")
```



Podemos alterar a aparência do raster que criamos usando o código seguinte.

```
>>> renderer = camadaR.renderer()
>>> provider = camadaR.dataProvider()
>>> stats = provider.bandStatistics(1, QgsRasterBandStats.All, rect, 0)
>>> min= stats.minimumValue
>>> max = stats.maximumValue
>>> range = max - min
>>> add = range//2
>>> interval = min + add
>>> colDic = {'re':'#ff0000', 'ye':'#ffff00', 'bl':'#0000ff'}
>>> valueList =[min, interval, max]
>>> lst = [QgsColorRampShader.ColorRampItem(valueList[0], Color(colDic['re'])),
QgsColorRampShader.ColorRampItem(valueList[1], QColor(colDic['ye'])),
QgsColorRampShader.ColorRampItem(valueList[2],QColor(colDic['bl']))]
>>> myRasterShader = QgsRasterShader()
>>> myColorRamp = QgsColorRampShader()
>>> myColorRamp.setColorRampItemList(lst)
>>> myRasterShader.setRasterShaderFunction(myColorRamp)
>>> myPseudoRenderer = QgsSingleBandPseudoColorRenderer(camadaR.dataProvider(),
camadaR.type(), myRasterShader)
>>> camadaR.setRenderer(myPseudoRenderer)
>>> camadaR.triggerRepaint()
```

O resultado da rampa de cores criada aplicado no raster será.



4. Executando python scripts fora do Qgis

Podemos executar processamentos do Qgis sem iniciar a interface gráfica, usando somente scripts. Para fazer isso temos que criar um script com a seguinte estrutura mínima.

```
from qgis.core import *
# indique onde o programa qgis está localizado no seu computador
QgsApplication.setPrefixPath("/usr", True)
# Crie uma referência à QgsApplication. Usando False como segundo argumento
# para não ativar a interface gráfica.
qgs = QgsApplication([], False)
# inicie o qgis
qgs.initQgis()
#####
# Escreva o código de processamento aqui##
#####
# finalize o script usando:
qgs.exitQgis()
```

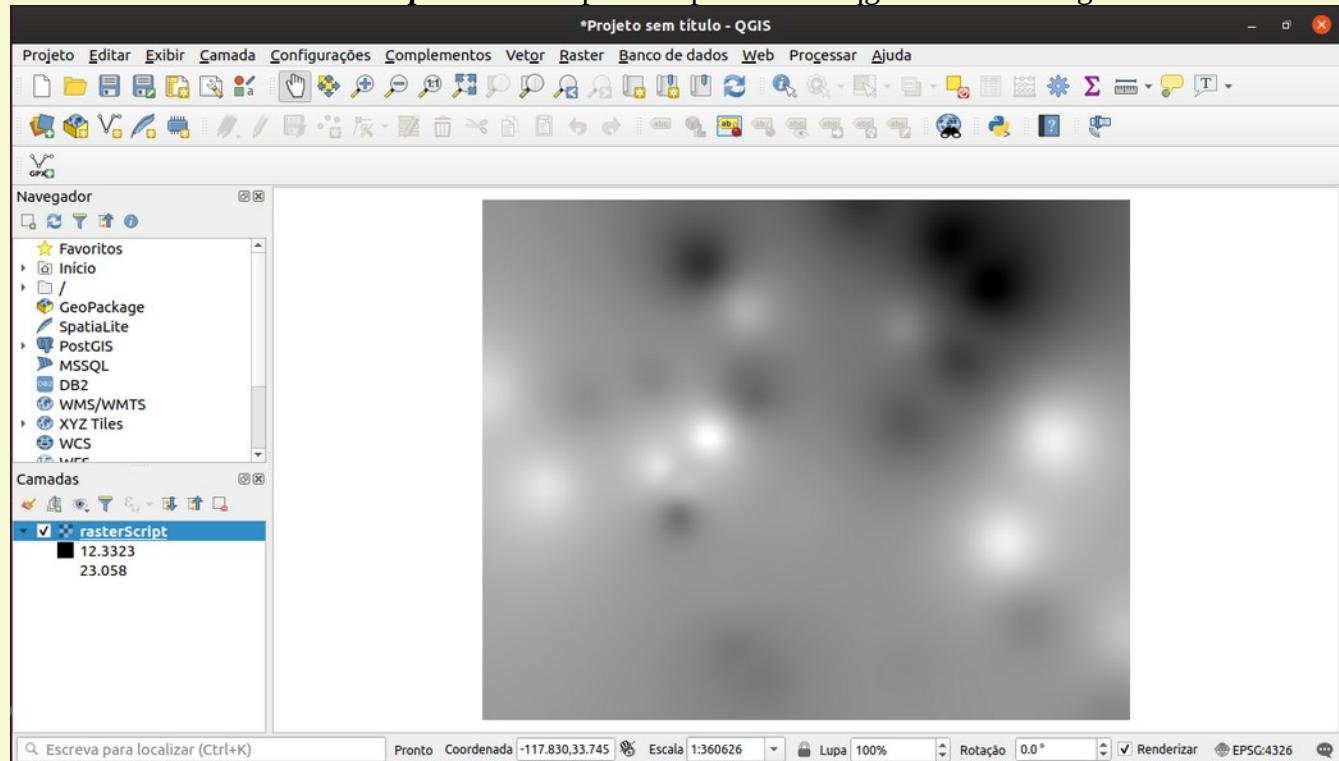
Vamos mostrar como isso funciona criando um script que criará um grid raster a partir de um arquivo texto CSV com alguns pontos. O mesmo procedimento do último exemplo do capítulo anterior. Nomeie o arquivo de **criaRaster.py**. Substitua /home/usr apropriadamente para o seu sistema.

```
from qgis.core import *
from qgis.analysis import *
QgsApplication.setPrefixPath("/usr", True)
qgs = QgsApplication([], False)
qgs.initQgis()
uri="file:///home/usr/tfinal.csv?
type=csv&xField=LONGITUDE&yField=LATITUDE&crs=epsg:4326"
camada = QgsVectorLayer(uri, 'Converte', "delimitedtext")
if not camada.isValid():
    print("A camada não carregou apropriadamente!")
else:
    c_data = QgsInterpolator.LayerData()
    c_data.source = camada
    c_data.zCoordInterpolation = False
    c_data.interpolationAttribute = 6
    c_data.sourceType = QgsInterpolator.SourcePoints
    interpolado = QgsIDWInterpolator([c_data])
    interpolado.setDistanceCoefficient(2)
    arquivo = "/home/usr/rasterScript.asc"
    rect = camada.extent()
    res = 0.001
    ncol = int( ( rect.xMaximum() - rect.xMinimum() ) / res )
    nrows = int( (rect.yMaximum() - rect.yMinimum() ) / res)
    saida = QgsGridFileWriter(interpolado,arquivo,rect,ncol,nrows)
    saida.writeFile()
qgs.exitQgis()
```

No ambiente linux/UNIX esse script funcionará sem problema usando **python3 criaRaster.py**, já para o ambiente Windows se faz necessário:

- Termos certeza que a variável PATH aponta para a pasta correta de instalação do Qgis, algo similar a e “C:/ProgramFiles/QGIS3.4/bin”
- Alterar a linha `QgsApplication.setPrefixPath("/usr", True)` para `QgsApplication.setPrefixPath("C:/ProgramFiles/QGIS3.4/bin/", True)`
- Executar usando **python-qgis-ltr criaRaster.py**

Ao abrirmos o raster **rasterScript.asc** criado pelo script acima no qgis veremos o seguinte.



No decorrer desta apostila faremos um uso intensivo de scripts fora do ambiente Qgis, tenha certeza que o exemplo mostrado acima funcionou perfeitamente no seu sistema.

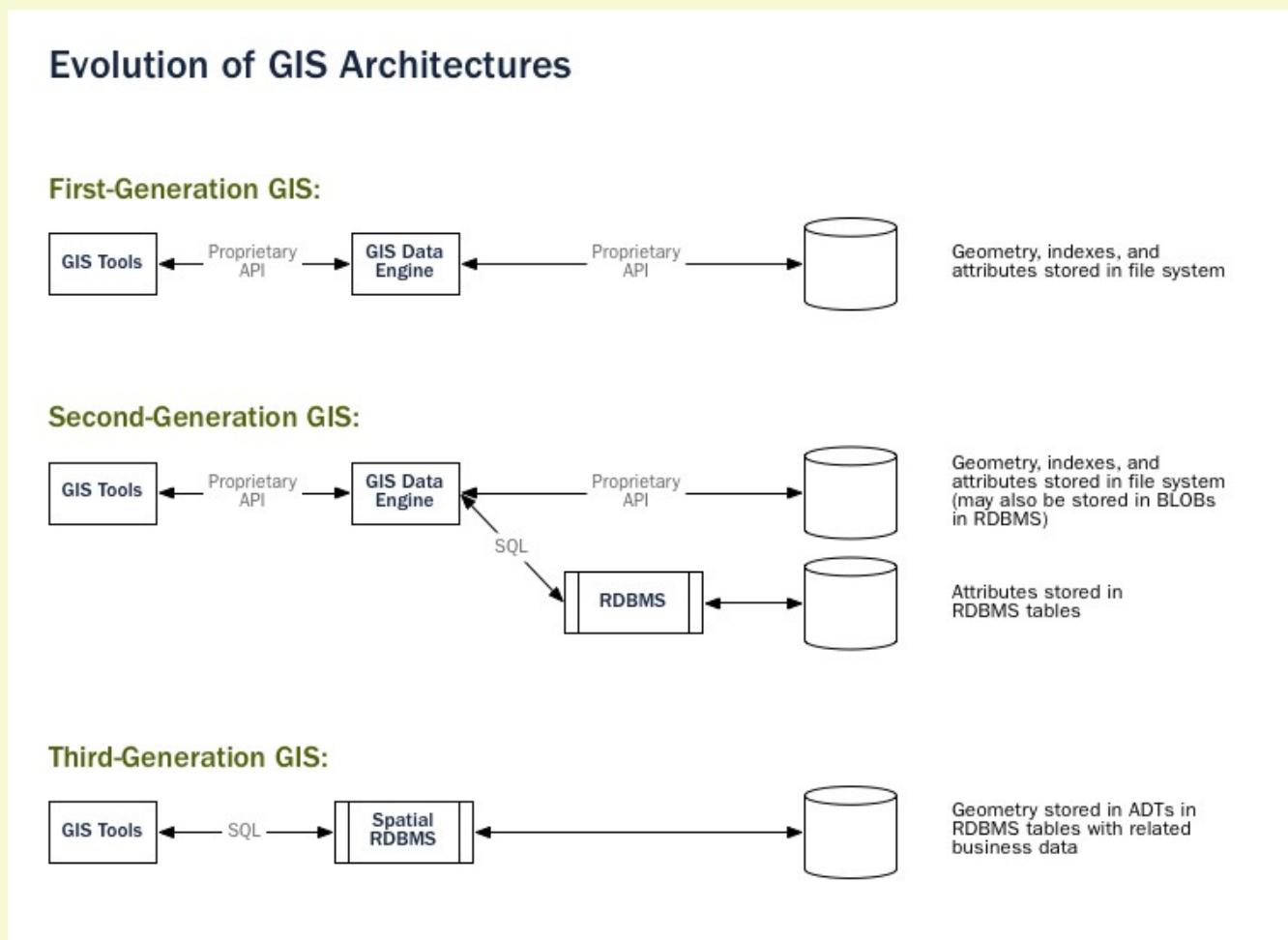
5. Integração PostgreSQL/Postgis com PyQgis

5.1 O Postgis

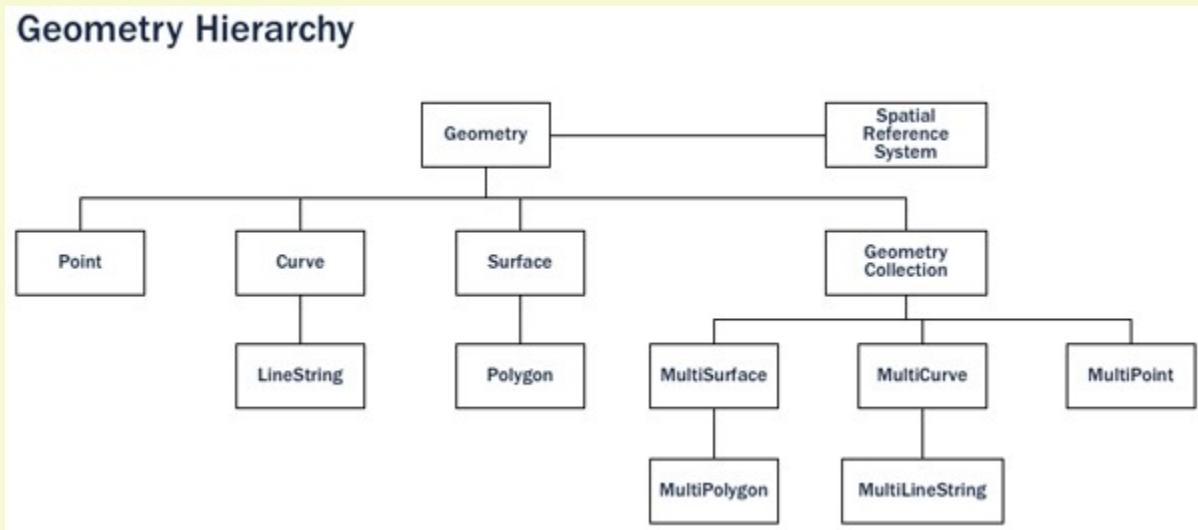
Postgis é um banco de dados espacial assim como o Oracle Spatial e o SQL Server (2008 em diante). O que faz um banco de dados comum se tornar em um banco de dados espacial é a capacidade de manipular qualquer objeto espacial deste banco de dados como se manipula um objeto comum (não espacial).

O Postgis transforma o sistema de gerenciamento do banco de dados PostgreSQL em um banco de dados espacial adicionando suporte a três componentes: tipo de dado espacial, índices e funções. O PostgreSQL é um sistema de banco de dados relacional (ORDBMS) poderoso e ‘open source’.

A figura abaixo brevemente descreve a evolução do GIS no contexto de um banco de dados espacial. A tendência hoje é que todos os dados estejam armazenados em bancos de dados espaciais e a ferramenta GIS, desktop ou web acesse diretamente este dado via SQL, sem a necessidade de software proprietário (GIS data engine) intermediando essa transação. Por isso é importante o conhecimento de SQL para todos aqueles que trabalham com GIS e geoprocessamento.



Tipos de dados espaciais se referem a formas (shapes) tais como pontos, linhas e polígonos; índices espaciais Multidimensionais são usados para processar operações de forma eficiente uma área destes dados espaciais; funções espaciais, nos termos de SQL são pesquisas (queries) de propriedades espaciais e relações entre elas. Combinados, estes três elementos (tipo de dados, índices e funções) nos dão uma estrutura flexível de análise e performance otimizadas.



5.2 Instalando PostgreSQL/Postgis

Vamos agora mostrar como montar um banco de dados espacial e como carregar dados dentro dele via pyQgis. Caso não exista a possibilidade ou interesse em montar um banco de dados pule para a sessão 2.6 desta apostila.

Escolha a forma de instalação compatível com o seu sistema operacional. Algumas versões podem já ter sido atualizadas.

5.2.1 – Windows

Do site www.enterprisedb.com/downloads/postgres-postgresql-downloads#windows selecione a versão do Postgresql e o sistema operacional windows (32 ou 64 bits).

EDB
POSTGRES

Products ▾ Cloud ▾ Customers ▾ Services and Support ▾ Training ▾ Resources ▾

SEARCH LOGIN MY ACCOUNT CONTACT

PostgreSQL 9.5.14

Windows x86-64

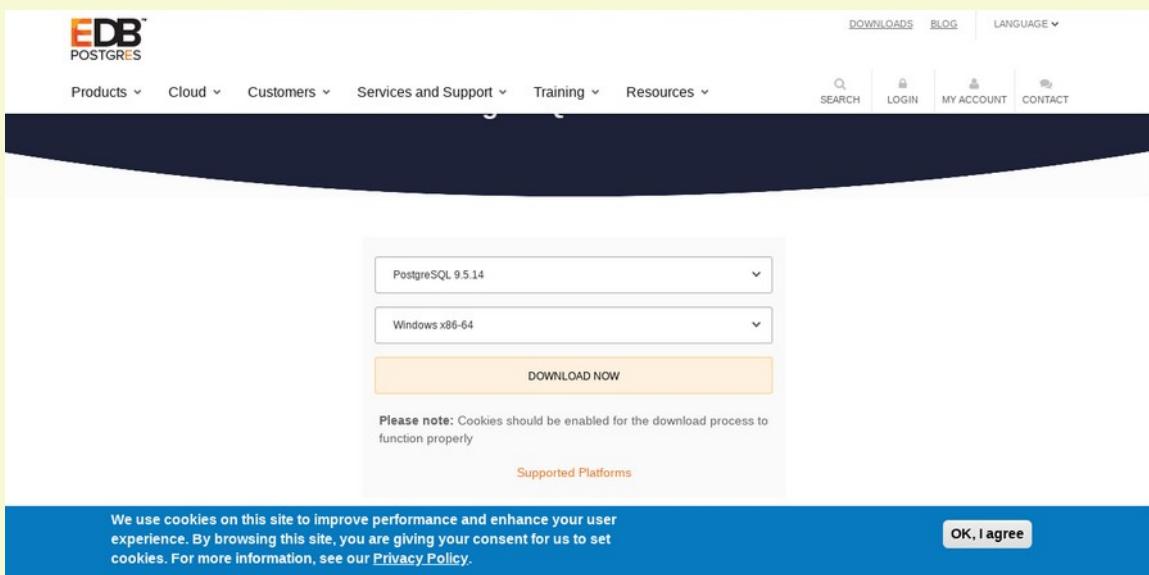
DOWNLOAD NOW

Please note: Cookies should be enabled for the download process to function properly

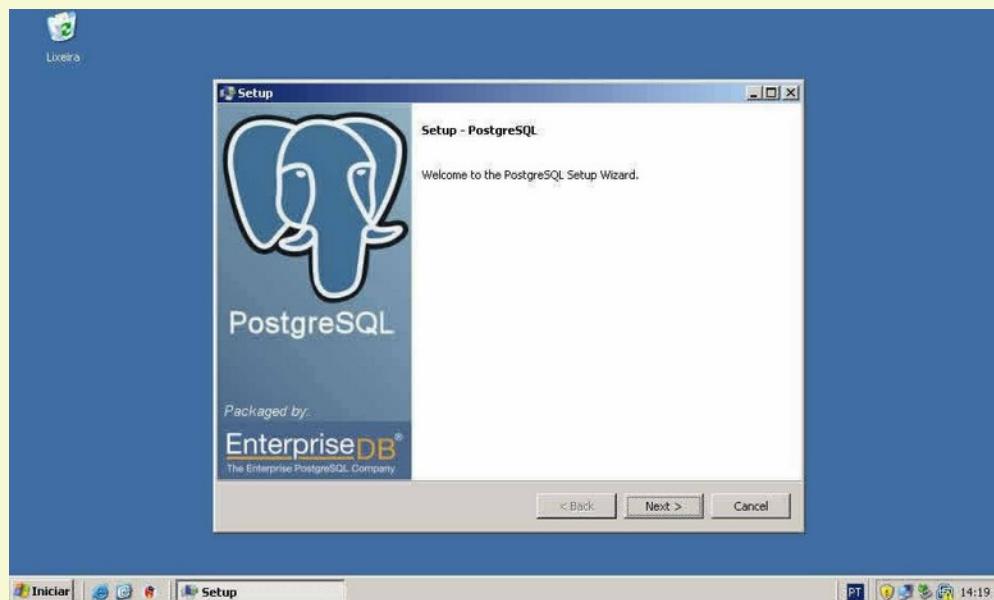
Supported Platforms

We use cookies on this site to improve performance and enhance your user experience. By browsing this site, you are giving your consent for us to set cookies. For more information, see our [Privacy Policy](#).

OK, I agree



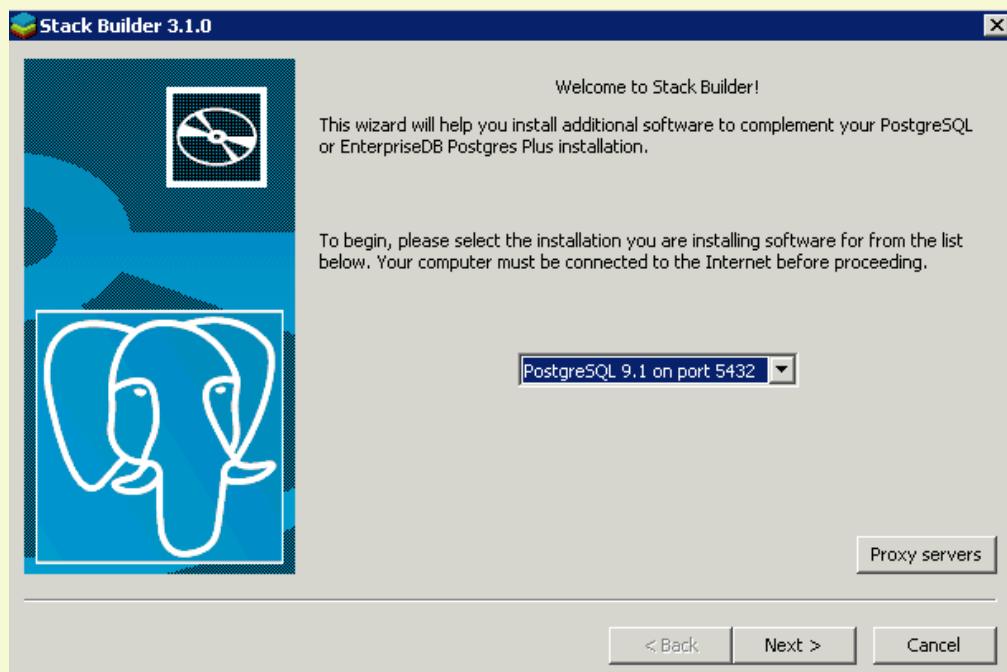
Faça o download e execute o .exe:



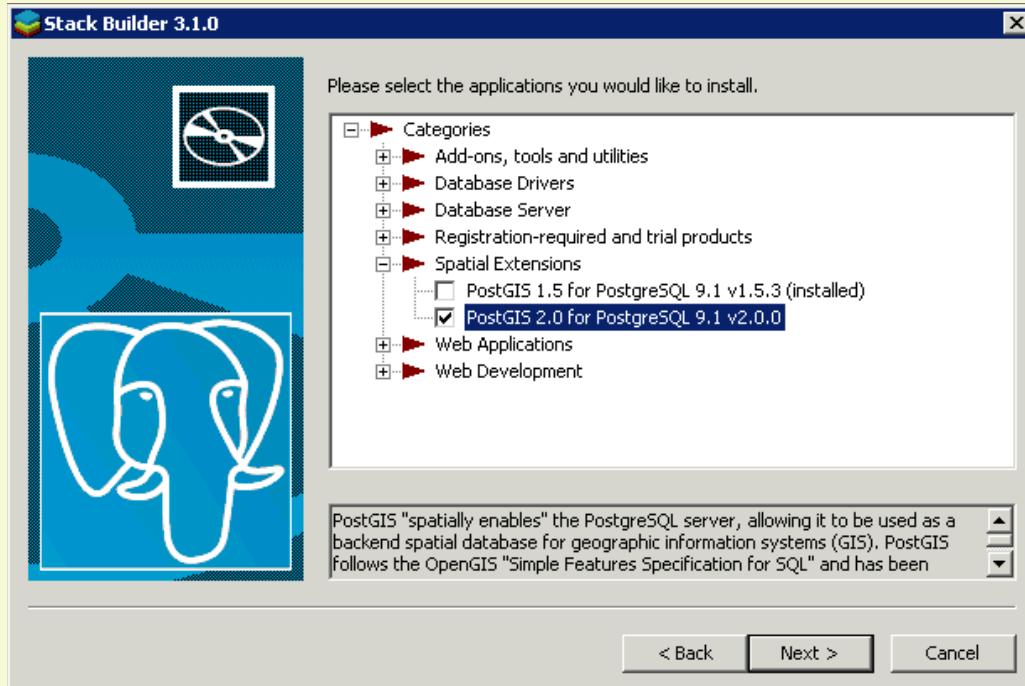
Siga as telas de instruções, selecionando as opções padrões, crie uma senha para o usuário postgres quando solicitado (guarde esta senha). Quando chegar na seguinte tela execute o StackBuilder para instalar o postgis.



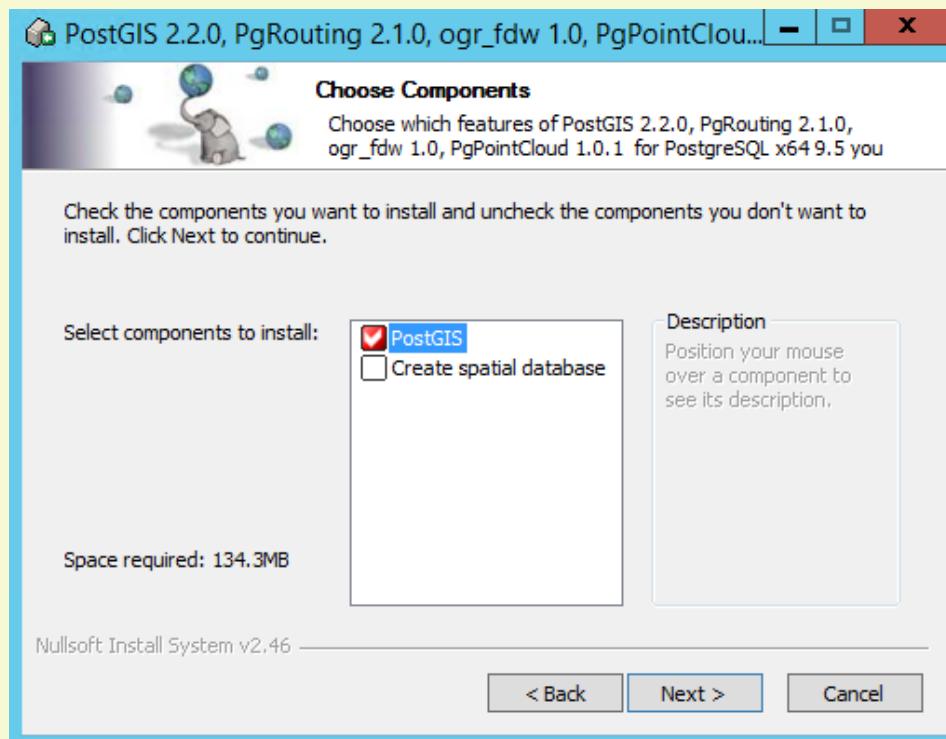
Uma vez instalada a versão do PostgreSQL execute o Stack Builder (Start → Programs → PostgreSQLX.X → Application StackBuilder).



Escolha o componente PostGIS



Prossiga sem checar a opção Create spatial database, quando perguntado para configurar variáveis de ambiente out-db-raster responda sim:



PostgreSQL e Postgis devem estar agora instalados.

5.2.2 - Linux - Centos ou RedHat (PostgreSQL e Postgis)

Abaixo segue um script de referência (copie e grave como pg_script.sh) para instalação num sistema RedHat ou Centos e execute usando `sudo ./pg_script.sh`

```
#!/bin/sh
# uso sudo bash pg_script.sh usuarioPG_a_criar senhaPG_a_criar
#- o usuário será super usuário do bd e deve ser um usuário do servidor também
#- senha diferente por favor!
user=$1
pass=$2
#-----pre- ferramentas-----
yum install -y epel-release
yum install -y perl
yum install -y wget
yum install -y unzip
yum install -y tar
#-----POSTGRESQL E POSTGIS-----
yum install -y https://download.postgresql.org/pub/repos/yum/9.5/redhat/rhel-7-x86_64/pgdg-redhat95-9.5-2.noarch.rpm
yum install -y postgresql95-server postgresql95-contrib postgresql95-devel
yum install -y geos36 geos36-devel
yum install -y gdal gdal-devel
yum install -y postgis2_95
/usr/pgsql-9.5/bin/postgresql95-setup initdb
systemctl enable postgresql-9.5.service
systemctl start postgresql-9.5.service
sed -i "\$a host all all 0.0.0.0/0 md5" /var/lib/pgsql/9.5/data/pg_hba.conf
sed -i "\$a listen_addresses = '*' /var/lib/pgsql/9.5/data/postgresql.conf
systemctl restart postgresql-9.5.service
su -i -u postgres << EOF
psql template1 -c "CREATE USER $user WITH PASSWORD $pass;""
psql template1 -c "ALTER USER $user WITH SUPERUSER;""
EOF
```

5.2.3 - OSX

A forma mais simples de instalar é usando homebrew. Instale homebrew conforme abaixo usando terminal:

```
$ xcode-select -install
$ /usr/bin/ruby -e "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/master/install)"
$ echo "export LC_ALL=en_US.UTF-8" >> ~/.bash_profile
$ echo "export LANG=en_US.UTF-8" >> ~/.bash_profile
$ echo "export PATH=/usr/local/bin:$PATH" >> ~/.bash_profile && source ~/.bash_profile
$
```

No terminal execute:

```
$ brew install postgres
$ brew install postgis
$ pg_ctl -D /usr/local/var/postgres start
$ initdb /usr/local/var/postgres
```

5.3 Criando um banco de dados Postgis

A primeira coisa que vamos fazer é montar nosso banco de dados com elementos mínimos para o funcionamento, ou seja, o banco de dados e extensões.

Tabelas serão implementadas usando pyQgis e nos servirão de base para o entendimento da estrutura de um banco de dados.

A extensão Postgis criará algumas tabelas adicionais para dar suporte a elementos do tipo raster e vetorial. Falaremos mais delas no decorrer deste guia. Execute os comandos abaixo no terminal monitor ou comando para criar o banco de dados (substitua **usuário** pelo seu nome de usuário).

```
$ createdb teste -O usuário --encoding=utf-8
$ psql -d teste -c "CREATE EXTENSION postgis;""
$ psql -d teste -c "ALTER DATABASE teste SET postgis.gdal_enabled_drivers TO 'GTiff PNG JPEG';"
```

Acabamos de criar um banco de dados ‘teste’. Nas seções seguintes veremos como interagir e carregar dados.

Um aspecto importante em se usar um banco de dados geoespacial é que sempre temos que estar atentos a qual sistema de referência de coordenadas (CRS) usaremos. Sempre utilize um único sistema de referência e de preferência use o WGS-84 sempre que possível.

5.4 Entrando dados no postgis

Uma vez que criamos o nosso banco de dados vamos ver como carregar dados dentro dele. O objetivo dessa apostila é o pyQgis e não o postgis, por esse motivo vamos focar em como usar o pyQgis para carregar, extrair e analisar os dados em um banco postgis. Não vamos cobrir extensivamente como o postgis funciona e nem falaremos sobre SQL, assumo que o leitor já tenha um conhecimento básico de SQL

Vamos utilizar o psycopg2 para interagir com o banco de dados PostgreSQL a partir do pyQgis. Instale ele usando o instalador de sua preferência (pip3,conda, etc).

5.4.1 Conectando com o banco de dados

O script **conecta.py** abaixo conectará com o banco de dados ‘teste’ e extrairá os parâmetros da conexão. Substitua **usr** e **snh** apropriadamente com a configuração de seu banco de dados.

```
#conecta.py
import psycopg2 as psy2
conn=psy2.connect(user="usr",password="snh",host="localhost",port="5432",database="teste")
cursor=conn.cursor()
print(conn.get_dsn_parameters(),"\n")
```

5.4.2 Inserindo dados vetoriais no banco de dados

Vamos adicionar os objetos espaciais dentro do banco de dados. Primeiramente mostraremos como criar tabelas via script e em seguida como criar tabelas a partir de outros formatos de dados (arquivos de gis, csv, etc). Faremos esse processo usando exemplos dos tipos Ponto (point), Linha (linestring) e Polígono (polygon).

Pontos

O script **adicPonto.py** abaixo criará uma tabela chamada cidades no banco de dados e em seguida criará um objeto vetorial do tipo ponto com três atributos, em seguida inserirá um elemento nele. Após a inserção carregará o conteúdo na tabela do banco de dados,

```
#adicPonto.py
# importando as libraries e iniciando o qgis
import psycopg2 as psy2
from qgis.core import *
from PyQt5.QtCore import QVariant
QgsApplication.setPrefixPath("/usr", True)
qgs = QgsApplication([], False)
qgs.initQgis()
# conectando e criando a tabela cidades
conn=psy2.connect(user="usr",password="snh",host="localhost",port="5432",database="teste")
cursor=conn.cursor()
cursor.execute("CREATE TABLE cidades (nome VARCHAR(32),populacao INT,idh numeric);")
cursor.execute("SELECT AddGeometryColumn ('public','cidades','geom',4326,
'POINT',2);")
vponto = QgsVectorLayer("Point?crs=EPSG:4326", "Cidades", "memory")
dPr = vponto.dataProvider()
dPr.addAttribute([QgsField("nome", QVariant.String), QgsField("populacao",
QVariant.Int), QgsField("idh", QVariant.Double)])
vponto.updateFields()
elem = QgsFeature()
elem.setGeometry(QgsGeometry.fromPointXY(QgsPointXY(-59.9936,-3.0925)))
elem.setAttributes(["Manaus", 2182763, 0.737])
dPr.addFeature(elem)
for ponto in vponto.getFeatures():
    geom = ponto.geometry()
    cursor.execute("INSERT INTO cidades (nome, populacao, idh, geom) VALUES (%s,
%s, %s, ST_GeomFromText(%s,4326));", (str(ponto['nome']),ponto['populacao'],
ponto['idh'], geom.asWkt()))
conn.commit()
qgs.exitQgis()
```

Agora vamos adicionar um arquivo de pontos do tipo shapefile dentro do banco de dados com o script **shpPonto.py** abaixo.

```
#shpPonto.py
# importando as libraries e iniciando o qgis
from qgis.core import *
QgsApplication.setPrefixPath("/usr", True)
qgs = QgsApplication([], False)
qgs.initQgis()
# entrando os parâmetros de conexão
nomeTabela = "temperatura"
```

```

uri = "dbname='teste' host=localhost port=5432 user='usr' password='snh'
type=POINT table=" + nomeTabela +" (geom)"
# carregando o arquivo shapefile a ser movido para o BD
camada = QgsVectorLayer("/home/usr/temper.shp", "temperatura", "ogr")
# executando a transferência
error = QgsVectorLayerExporter.exportLayer(camada, uri, "postgres", camada.crs(), False)
if error[0] != 0:
    print('Erro', error[1])
# finalizando qgis
qgs.exitQgis()

```

Linhas

Criaremos uma tabela de objetos do tipo linha no banco de dados de forma similar ao que fizemos com pontos. O script **adicLinha.py** abaixo fará isso.

```

#adicLinha.py
# importando as libraries e iniciando o qgis
import psycopg2 as psy2
from qgis.core import *
from PyQt5.QtCore import QVariant
QgsApplication.setPrefixPath("/usr", True)
qgs = QgsApplication([], False)
qgs.initQgis()
# conectando e criando a tabela rota
conn=psy2.connect(user="usr",password="snh",host="localhost",port="5432",database="teste")
cursor=conn.cursor()
cursor.execute("CREATE TABLE rota (nome VARCHAR(32),numero INT);")
cursor.execute("SELECT AddGeometryColumn ('public','rota','geom',4326,
'LINESTRING',2);")
vlinha = QgsVectorLayer("Linestring?crs=EPSG:4326", "Vias", "memory")
dPr = vlinha.dataProvider()
dPr.addAttribute([QgsField("nome", QVariant.String), QgsField("numero",
QVariant.Int)])
vlinha.updateFields()
elem = QgsFeature()
pontos =[QgsPoint(-60.049,-3.038), QgsPoint(-59.131,-3.600 ), QgsPoint(-59.607,-
4.406),QgsPoint(-60.369,-5.119)]
elem.setGeometry(QgsGeometry.fromPolyline(pontos))
elem.setAttributes(["Rota ", 1])
dPr.addFeature(elem)
for linha in vlinha.getFeatures():
    geom = linha.geometry()
    cursor.execute("INSERT INTO rota (nome, numero, geom) VALUES (%s, %s,
ST_GeomFromText(%s,4326));", (str(linha['nome']),linha['numero'], geom.asWkt()))
conn.commit()
qgs.exitQgis()

```

Agora vamos adicionar um objeto linha do tipo banco de dados postgis remoto dentro do banco de dados postgis local com o script **pgLinha.py** abaixo.

```

#pgLinha.py
# importando as libraries e iniciando o qgis
from qgis.core import *

```

```

QgsApplication.setPrefixPath("/usr", True)
qgs = QgsApplication([], False)
qgs.initQgis()
# entrando os parâmetros de conexão
nomeTabela = "gasoduto"
uri = "dbname='teste' host=localhost port=5432 user='usr' password='snh'"
type=LINESTRING table=" + nomeTabela +" (geom)"
# carregando o arquivo shapefile a ser movido para o BD
uri2 = QgsDataSourceUri()
uri2.setConnection("amazeone.com.br","5432","dnpm","droid", "devcor")
uri2.setDataSource("public", "duto", "geom")
camada = QgsVectorLayer(uri2.uri(False), "gasoduto", "postgres")
# executando a transferência
error = QgsVectorLayerExporter.exportLayer(camada, uri, "postgres", camada.crs(), False)
if error[0] != 0:
    print('Erro', error[1])
# finalizando qgis
qgs.exitQgis()

```

Polígonos

Agora criaremos uma tabela de objetos do tipo Polígono no banco de dados. Isto é feito usando o script **adicPoligono.py** abaixo.

```

#adicPoligono.py
# importando as libraries e iniciando o qgis
import psycopg2 as psy2
from qgis.core import *
from PyQt5.QtCore import QVariant
QgsApplication.setPrefixPath("/usr", True)
qgs = QgsApplication([], False)
qgs.initQgis()
# conectando e criando a tabela rota
conn=psy2.connect(user="usr",password="snh",host="localhost",port="5432",database="teste")
cursor=conn.cursor()
cursor.execute("CREATE TABLE area_tur (nome VARCHAR(32),numero INT);")
cursor.execute("SELECT AddGeometryColumn ('public','area_tur','geom',4326,
'POLYGON',2);")
vpol = QgsVectorLayer("Polygon?crs=EPSG:4326", "Area", "memory")
dPr = vpol.dataProvider()
dPr.addAttribute([QgsField("nome", QVariant.String), QgsField("numero",
QVariant.Int)])
vpol.updateFields()
elem = QgsFeature()
pontos =[[QgsPointXY(-61.19,-2.11), QgsPointXY(-60.74,-2.02 ), QgsPointXY(-60.36,-
3.02),QgsPointXY(-60.58,-3.02)]]
elem.setGeometry(QgsGeometry.fromPolygonXY(pontos))
elem.setAttributes(["Anavilhanas ", 1])
dPr.addFeature(elem)
for poligono in vpol.getFeatures():
    geom = poligono.geometry()
    cursor.execute("INSERT INTO area_tur (nome, numero, geom) VALUES(%s,%s,ST_GeomFromText(%s,4326));", (str(poligono['nome']),poligono['numero'],
geom.asWkt()))
    conn.commit()
qgs.exitQgis()

```

Agora vamos adicionar um arquivo de polígono do tipo shapefile dentro do banco de dados com o script **shpPolig.py** abaixo.

```
#shpPolig.py
# importando as libraries e iniciando o qgis
from qgis.core import *
QgsApplication.setPrefixPath("/usr", True)
qgs = QgsApplication([], False)
qgs.initQgis()
# entrando os parâmetros de conexão
nomeTabela = "amazonas"
uri = "dbname='teste' host=localhost port=5432 user='usr' password='snh'"
type=POLYGON table=" + nomeTabela +" (geom)"
# carregando o arquivo shapefile a ser movido para o BD
camada = QgsVectorLayer("/home/usr/amazonas.shp","limites", "ogr")
# executando a transferência
error = QgsVectorLayerExporter.exportLayer(camada, uri, "postgres", camada.crs(), False)
if error[0] != 0:
    print('Erro', error[1])
# finalizando qgis
qgs.exitQgis()
```

5.4.3 Inserindo dados raster no banco de dados

Existem duas formas de carregar uma imagem raster no banco de dados postgis, vamos ver agora a maneira mais simples usando o programa raster2pgsql que vem junto com o Postgis. Primeiro vamos repetir o script onde criamos o arquivo raster e em seguida vamos executar o raster2pgsql via python **os** library. Chamaremos esse script de **adicRaster.py**.

```
# adicRaster.py
from qgis.core import *
from qgis.analysis import *
QgsApplication.setPrefixPath("/usr", True)
qgs = QgsApplication([], False)
qgs.initQgis()
uri="file:///home/usr/tfinal.csv?
type=csv&xField=LONGITUDE&yField=LATITUDE&crs=epsg:4326"
camada = QgsVectorLayer(uri, 'Converte', "delimitedtext")
if not camada.isValid():
    print("A camada não carregou apropriadamente!")
else:
    c_data = QgsInterpolator.LayerData()
    c_data.source = camada
    c_data.zCoordInterpolation = False
    c_data.interpolationAttribute = 6
    c_data.sourceType = QgsInterpolator.SourcePoints
    interpolado = QgsIDWInterpolator([c_data])
    interpolado.setDistanceCoefficient(2)
    arquivo = "/home/usr/rasterScript.asc"
    rect = camada.extent()
    res = 0.001
    ncol = int( ( rect.xMaximum() - rect.xMinimum() ) / res )
    nrows = int( (rect.yMaximum() - rect.yMinimum() ) / res)
    saida = QgsGridFileWriter(interpolado,arquivo,rect,ncol,nrows)
    saida.writeFile()
```

```

qgs.exitQgis()
import os
os.system("raster2pgsql -s 4326 -t 50x50 -d -C -I -Y rasterScript.asc
public.temperaster | psql -d teste ")

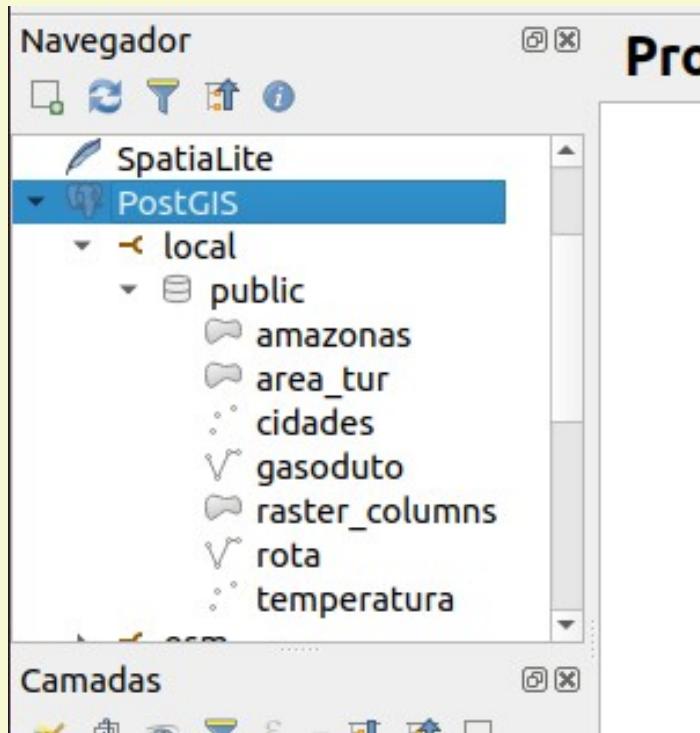
```

Na Parte 3 veremos como interagir e carregar raster da outra forma citada, usando SQL puro.

5.5 Visualizando os dados criados

Com os dados vetoriais e raster criados e carregados no banco de dados podemos visualizar eles usando o Qgis.

Criamos uma nova conexão postgis no navegador para o host localhost e banco de dados teste, daremos o nome de local para esse conector postgis. Apos criar, clique nele e as credenciais de conexão serão pedidas, entre com o usuário e senha e as tabelas deverão aparecer conforme a imagem abaixo após clicar em public.

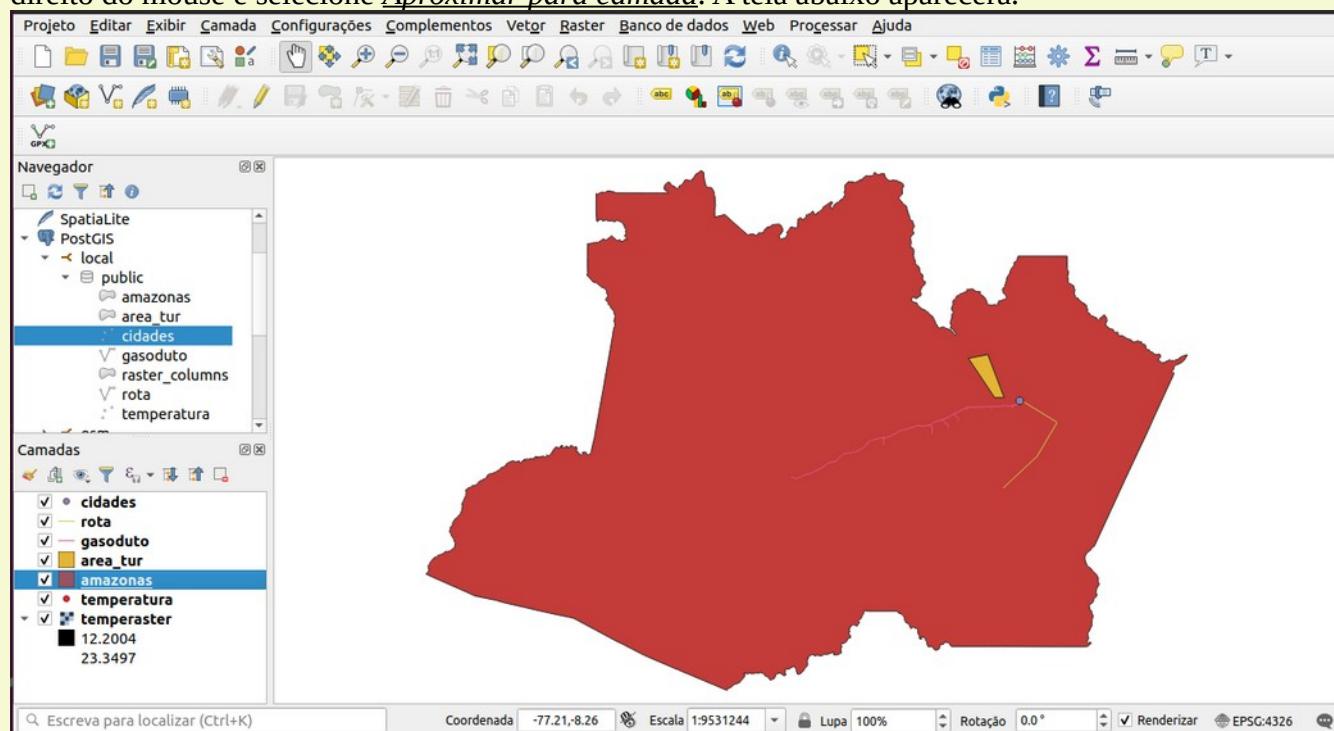


Carregaremos agora a imagem raster. Isso é feito usando o menu Banco de Dados > Gerenciador de BD. Nele selecionamos Postgis e local e entramos a senha. Clicamos em public e clicamos com o botão direito em **temperaster** e selecionamos adicionar a tela. O raster será carregado na tela. Feche o gerenciador DB.

No navegador a esquerda clique duas vezes na camada temperatura e a camada de pontos será carregada sobre a imagem raster.

O resultado será algo semelhante ao mostrado abaixo.

Vamos agora ver as demais camadas, clique duas vezes na camada *amazonas* no navegador, também em *area_tur*, em *gasoduto*, em *rota* e em *cidades*. Em camadas selecione *amazonas*, clique com o botão direito do mouse e selecione Aproximar para camada. A tela abaixo aparecerá.



5.6 Extraindo dados do Postgis usando pyQgis

Vamos agora trabalhar com dados existentes em um banco de dados remoto que contém os processos minerários atualizados do Estado do Tocantins. Vamos trabalhar com essa informação usando queries (seleção por perguntas) ao banco de dados baseados em parâmetros/atributos dos dados e/ou parâmetros espaciais.

5.6.1 Conexão ao banco de dados e tabelas usadas

Primeiramente conectaremos ao banco de dados usando a seguinte “string de conexão”.

```
user="droid",password="devcor",host="amazone.com.br",port="5432",database="dnpmto"
```

Trabalharemos com as seguintes tabelas deste banco de dados:

sig2 (dados dos processos minerários)

Coluna	Tipo
processo	text
fase	text
ult_evento	text
nome	text
subs	text
data	date
geom	geometry(MultiPolygon,4326)

indio (reservas indígenas)

Coluna		Tipo
ID		text
CD_GEOCODM		text
NM_MUNICIP		text
gid		integer
terrai_cod		integer
terrai_nom		text
etnia_nome		text
municipio_		text
uf_sigla		text
superficie		double precision
fase_ti		text
modalidade		text
reestudo_t		text
cr		text
faixa_fron		text
undadm_cod		double precision
undadm_nom		text
undadm_sig		text
geom		geometry(Polygon,4326)

ucpi (unidade de conservação de proteção integral)

Coluna		Tipo
ID		text
CD_GEOCODM		text
NM_MUNICIP		text
ID_UC0		text
NOME_UC1		text
ID_WCMC2		text
CATEGORI3		text
GRUPO4		text
ESFERA5		text
ANO_CRIA6		text
GID7		text
QUALIDAD8		text
ATO_LEGA9		text
DT_ULTIM10		text
CODIGO_U11		text
NOME_ORG12		text
geom		geometry(MultiPolygon,4326)

ucus (unidade de conservação de uso sustentável)

Coluna		Tipo
ID		text
CD_GEOCODM		text

NM_MUNICIP	text
ID_UC0	text
NOME_UC1	text
ID_WCMC2	text
CATEGORI3	text
GRUPO4	text
ESFERA5	text
ANO_CRIA6	text
GID7	text
QUALIDAD8	text
ATO_LEGA9	text
DT_ULTIM10	text
CODIGO_U11	text
NOME_ORG12	text
geom	geometry(MultiPolygon,4326)

O script **colunas.py** abaixo conecta ao banco de dados e mostra as colunas da tabela. Select * seleciona todas as colunas da tabela e usamos LIMIT 0 para somente obter a descrição da tabela evitando a transferência de informações que não necessitamos nesse caso.

```
#colunas.py
import psycopg2 as psy2
conn=psy2.connect(user="droid",password="devcor",host="amazeone.com.br",port="5432",
,database="dnpmto")
cursor=conn.cursor()
tabelas=['sig2','indio','ucus','ucpi']
for tabela in tabelas:
    cursor.execute("Select * FROM "+tabela+ " LIMIT 0")
    colnomes = [desc[0] for desc in cursor.description]
    print("\n"+tabela)
    print(colidomes)
```

Resultado:

```
sig2
['processo', 'fase', 'ult_evento', 'nome', 'subs', 'data', 'geom']

indio
['ID', 'CD_GEOCODM', 'NM_MUNICIP', 'gid', 'terrai_cod', 'terrai_nom',
'etnia_nome', 'municipio_', 'uf_sigla', 'superficie', 'fase_ti',
'modalidade', 'reestudo_t', 'cr', 'faixa_fron', 'undadm_cod',
'undadm_nom', 'undadm_sig', 'geom']

ucus
['ID', 'CD_GEOCODM', 'NM_MUNICIP', 'ID_UC0', 'NOME_UC1', 'ID_WCMC2',
'CATEGORI3', 'GRUPO4', 'ESFERA5', 'ANO_CRIA6', 'GID7', 'QUALIDAD8',
'ATO_LEGA9', 'DT_ULTIM10', 'CODIGO_U11', 'NOME_ORG12', 'geom']

ucpi
['ID', 'CD_GEOCODM', 'NM_MUNICIP', 'ID_UC0', 'NOME_UC1', 'ID_WCMC2',
'CATEGORI3', 'GRUPO4', 'ESFERA5', 'ANO_CRIA6', 'GID7', 'QUALIDAD8',
'ATO_LEGA9', 'DT_ULTIM10', 'CODIGO_U11', 'NOME_ORG12', 'geom']
```

5.6.2 Obtendo valores distintos de uma coluna da tabela

Muitas vezes precisaremos saber os valores distintos de uma tabela para fazermos uma seleção. No script **distinto.py** abaixo vamos extrair os valores distintos da coluna **fase** da tabela **sig2**.

```
#distinto.py
import psycopg2 as psy2
conn=psy2.connect(user="droid",password="devcor",host="amazeone.com.br",port="5432",
,database="dnpmto")
cursor=conn.cursor()
cursor.execute("Select distinct fase FROM sig2")
linhas = cursor.fetchall()
i=1
for linha in linhas:
    print(" Valor "+str(i)+": ", linha[0])
    i=i+1
```

Resultado:

```
Valor 1: CONCESSÃO DE LAVRA
Valor 2: REGISTRO DE EXTRAÇÃO
Valor 3: REQUERIMENTO DE PESQUISA
Valor 4: LAVRA GARIMPEIRA
Valor 5: REQUERIMENTO DE LAVRA
Valor 6: DISPONIBILIDADE
Valor 7: AUTORIZAÇÃO DE PESQUISA
Valor 8: REQUERIMENTO DE REGISTRO DE EXTRAÇÃO
Valor 9: REQUERIMENTO DE LICENCIAMENTO
Valor 10: LICENCIAMENTO
Valor 11: DIREITO DE REQUERER A LAVRA
Valor 12: REQUERIMENTO DE LAVRA GARIMPEIRA
```

5.6.3 Usando a cláusula WHERE

Vamos agora mostrar no script **onde.py** como podemos facilmente selecionar registros usando a cláusula WHERE do SQL

```
#onde.py
import psycopg2 as psy2
conn=psy2.connect(user="droid",password="devcor",host="amazeone.com.br",port="5432",
,database="dnpmto")
cursor=conn.cursor()
cursor.execute("Select nome,processo FROM sig2 WHERE fase='CONCESSÃO DE LAVRA'")
linhas = cursor.fetchall()
for linha in linhas:
    print(linha[1], linha[0])
```

Resultado

```
864176/2004 Itafos Arraias Mineração e Fertilizantes S.a.
864262/2001 DRAGA MINAS EXTRAÇÃO DE PEDRAS LTDA
860128/1983 Rio Novo Mineração Ltda.
861274/1986 Terra Goyana Mineradora Ltda
864513/2006 Nativa Mineração Ltda
864148/2003 Caltins Calcário Tocantins Ltda
860787/1993 Mineração Rodolita Ltda
```

864431/2012 Vereda Ltda
864080/2001 MINERAÇÃO CAPITAL LTDA. "ME"
860266/1993 Vecon Construtora e Incorporadora Ltda
860232/1990 Engegold Mineração Ltda
861861/1984 Gessonorte Ind e Com de Produtos de Mineracao Ltda Epp
864091/1996 IGUATU WATERS LTDA EPP
864197/2004 Empresa de Mineração Floresta Negra Ltda.
861293/1991 Pedreiras Paraíso Ltda
864173/2004 Itafos Arraias Mineração e Fertilizantes S.a.
864175/2004 Itafos Arraias Mineração e Fertilizantes S.a.
864105/2009 Rialma Fertilizantes Indústria e Comércio S A
864258/2001 MINERAÇÃO CAPITAL LTDA. "ME"
864037/2004 Aliança Indústria de Bebidas e Alimentos Ltda
7722/1967 Gessonorte Ind e Com de Produtos de Mineracao Ltda Epp
864523/2006 Nativa Mineração Ltda
864352/1995 Rio dos Mangues Mineração Ltda
864174/2004 Itafos Arraias Mineração e Fertilizantes S.a.
860399/1991 EMPRESA SUL AMERICANA DE MONTAGENS S/A
860276/1992 lh engenharia e mineração ltda
860209/1993 Água Santa Clara Indústria e Comércio de Bebidas Ltda
860248/1989 Pedreira Anhanguera S A Empresa de Mineração
860634/1988 Pedreiras Paraíso Ltda
860247/1989 Pedreira Anhanguera S A Empresa de Mineração
864263/2001 MINERAÇÃO CAPITAL LTDA. "ME"
864147/2000 Mito Mineração Tocantins Ltda. Me
806742/1975 Calcário Cristalândia Ltda.
864098/2001 Mito Mineração Tocantins Ltda. Me
862224/1980 Rio Novo Mineração Ltda.
860933/1982 Votorantim Cimentos N Ne S A
864104/2001 INDUSTRIAL BRITAGEM CONCRETO E TRANSPORTE LTDA
864417/1996 Votorantim Cimentos N Ne S A
807131/1977 Mineração Rio Formoso Ltda
864259/2001 MINERAÇÃO CAPITAL LTDA. "ME"
864353/1995 Rio dos Mangues Mineração Ltda
860757/1990 Mineração São Francisco Ltda
812913/1976 Mineração Rio Formoso Ltda
864205/2013 Pires e Camargo Ltda. Epp
864154/2009 Rialma Fertilizantes Indústria e Comércio S A
864354/1995 Rio dos Mangues Mineração Ltda
800708/1977 Best Metais e Soldas S.A.
860327/1993 Coleme Mineração Ltda
864178/2003 Votorantim Cimentos N Ne S A
864501/2008 Daqui Agroindústria Importação e Exportação Ltda
864113/2003 Itafos Arraias Mineração e Fertilizantes S.a.
864053/1998 Marconcelos Mineração Ltda.
864455/2010 Agua Mineral Satisfaz Eireli Epp
864037/2002 Caltins Calcário Tocantins Ltda
864000/1998 Mito Mineração Tocantins Ltda. Me
864147/2001 Mito Mineração Tocantins Ltda. Me

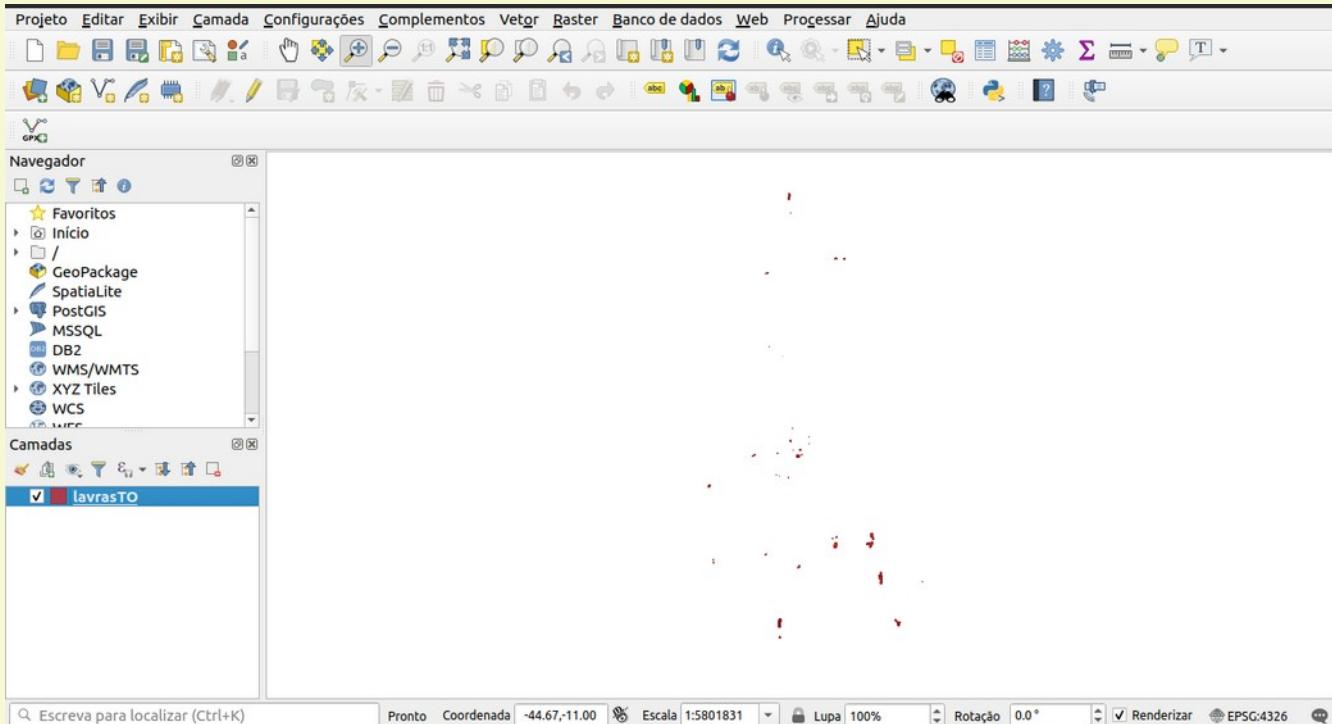
5.6.4 Selecionando dados espaciais e criando arquivo do resultado

Finalizaremos esta parte gerando um arquivo gis (shapefile) localmente com base em uma seleção filtrada de uma tabela no banco de dados. Vamos criar o arquivo com os processos que são concessão de lavra no estado do Tocantins no script **lavra.py** abaixo.

```
# lavra.py
from qgis.core import *
QgsApplication.setPrefixPath("/usr", True)
qgs = QgsApplication([], False)
qgs.initQgis()
uri = QgsDataSourceUri()
uri.setConnection('amazeone.com.br', '5432', 'dnpmto', 'droid', 'devcor')
uri.setDataSource('public','sig2','geom',"fase='CONCESSÃO DE LAVRA'")
camada = QgsVectorLayer(uri.uri(False), 'Concessao', 'postgres')
QgsVectorFileWriter.writeAsVectorFormat(camada, '/home/andre/curso/lavrasTO.shp',
'utf-8', driverName='ESRI Shapefile')
qgs.exitQgis()
```

O método **setConnection()** usa os parâmetros ('servidor', 'porta', 'banco de dados', 'usuário', 'senha') e o método **setDataSource()** usa como parâmetro ('esquema da tabela', 'nome da tabela', 'coluna com a geometria', 'cláusula WHERE')

Ao abrirmos o arquivo **lavrasTO.shp** recém-criado no Qgis veremos:



6. Introdução à análise espacial no pyQgis

6.1 O conceito de análise espacial

Podemos de forma simples definir Análise Espacial como sendo a correlação/distribuição/variação de certo tipo de dados vinculada a sua posição espacial num plano bi (xy), tri (xyz) ou tetra (xyz e tempo) dimensional. Qual é a correlação de determinada(s) grandeza(s) no espaço (e porque não, no tempo)?

6.1.1 Exemplo 1 – A primeira análise espacial

John Snow, conhecido como o primeiro analista geoespacial, produziu em 1854 o seu famoso mapa mostrando o número e a localização das mortes causadas por um surto de cólera no bairro de Soho em Londres. Neste mapa ele também mostra a localização das bombas d'água do bairro.



Vamos fazer um exercício para identificar qual bomba poderia ser a fonte da contaminação.

Primeiro criaremos um buffer com um raio de 250 metros de cada bomba d'água. Depois vamos checar quantas mortes por cólera ocorreram dentro de cada buffer. De forma complementar, podemos

investigar de quais direções todas as mortes ocorreram com relação à bomba d'água. Se os casos ocorrem em todas as direções de uma bomba, esta estará próxima do foco ou é o próprio foco de contaminação.

Preparando

Os dados dos poços d'água e das mortes estão carregados no banco de dados **jsnow** no servidor **amazeone.com.br** acesse com o usuário **droid** e senha **devcor**. Caso queira criar um banco de dados localmente, baixe os dados no repositório e siga o procedimento abaixo.

1 - Crie o banco de dados usando

```
createdb jsnow -O usr --encoding=utf-8
psql -d jsnow -c "CREATE EXTENSION postgis;"
psql -d jsnow -c "ALTER DATABASE jsnow SET postgis.gdal_enabled_drivers TO 'GTiff PNG JPEG';"
```

2 - Baixe os arquivos mortescolera.* , bombasdagua.* e mapaJSnow.tif. Fonte dos dados no blog de Robin Wilson (<http://blog.rtwilson.com/john-snows-famous-cholera-analysis-data-in-modern-gis-formats/>).

3 - Execute o script **jsnow.py** abaixo, ajuste o caminho para os aquivos e os valores de **usr** e **snh**.

```
#jsnow.py
from qgis.core import *
QgsApplication.setPrefixPath("/usr", True)
qgs = QgsApplication([], False)
qgs.initQgis()
tabelas=['mortescolera', 'bombasdagua']
for tabela in tabelas:
    uri = "dbname='jsnow' host=localhost port=5432 user='usr' password='snh'"
    type=MULTIPOINT table)+"tabela" "(geom)"
    camada = QgsVectorLayer("/home/usr/" + tabela + ".shp", tabela, "ogr")
    error = QgsVectorLayerExporter.exportLayer(camada, uri, "postgres",
    camada.crs(), False)
    if error[0] != 0:
        print('Erro', error[1])
qgs.exitQgis()
```

Executando

Seja usando o banco de dados remoto ou usando o seu banco de dados conforme criado acima, execute o seguinte script **jsnowanalise.py** para obter o resultado da análise espacial conforme planejamos. Instale as libraries (psycopg2, windrose, ,matplotlib e numpy) usando pip ou conda caso estas ainda não existam na sua instalação de python.

```
#jsnowanalise.py
# carregando as libraries necessárias-----
import psycopg2 as psy2 # library conexão com postgresql
from windrose import WindroseAxes # library gráficos do tipo rosa dos ventos
from matplotlib import pyplot as plt # library gráficos
import matplotlib.gridspec as gridspec # library grid de gráficos
import numpy as np # library numérica
# conectando com o banco de dados usando um cursor-----
conn=psy2.connect(user="usr",password="snh",host="hs",port="5432",database="jsnow")
cursor=conn.cursor()
#executando o query via cursor (ver Nota abaixo)
cursor.execute("SELECT b.id,sum(a.count),ARRAY(SELECT
degrees(ST_Azimuth(st_geometryN(bombasdagua.geom,1),
```

```

st_geometryN(mortescolera.geom,1))) as deg FROM bombasdagua,mortescolera WHERE
bombasdagua.id=b.id AND ST_Dwithin(bombasdagua.geom, mortescolera.geom, 250)) FROM
mortescolera a, bombasdagua b WHERE ST_Dwithin(a.geom, b.geom, 250) GROUP BY
b.id;")
# carregando o resultado do query em linhas
linhas = cursor.fetchall()
# preparando o grid de gráficos e executando loop nos resultados-----
fig = plt.figure(figsize=(18, 12)) # tamanho do plot
gs = gridspec.GridSpec(2, 4) # 2 linhas 4 colunas
base, topo, esq, dire = gs.get_grid_positions(fig) # coordenadas de cada plot
l=0 # identificador do plot rosa dos ventos
for linha in linhas: #loop que cria cada rosa dos ventos com soma de mortes
    bomba=linha[0]
    mortes=linha[1]
    lin, col = (l//4)%3 , l%4
    ret = [esq[col],base[lin],dire[col]-esq[col],0.9*(topo[lin]-base[lin])]
    rosa = WindroseAxes(fig, ret)
    fig.add_axes(rosa)
    angulos=linha[2]
    quantidade = [1]*len(angulos)
    rosaQ = np.array(quantidade)
    rosaA = np.array(angulos)
    rosa.box(rosaA, rosaQ)
    plt.title("Bomba:"+str(bomba)+ " Mortes:"+str(mortes), pad=12.)
    l=l+1
# plotando o gráfico final
plt.show()

```

Nota:

O query abaixo que foi usado no script para extrair a informação que precisamos nesta análise espacial.

```

SELECT
    b.id,
    sum(a.count),
    ARRAY(SELECT
        degrees(ST_Azimuth(ST_GeometryN(bombasdagua.geom,1),
        ST_GeometryN(mortescolera.geom,1))) AS deg
    FROM bombasdagua,mortescolera
    WHERE bombasdagua.id=b.id AND ST_Dwithin(bombasdagua.geom,
        mortescolera.geom, 250))
FROM mortescolera a, bombasdagua b
WHERE ST_Dwithin(a.geom, b.geom, 250)
GROUP BY b.id;

```

Que é traduzido como:

Ele extrai (SELECT) o

*id da tabela **bombasdagua**,*

*a soma (sum) das mortes da tabela **mortescolera** agrupada por id (GROUP BY b.id),*

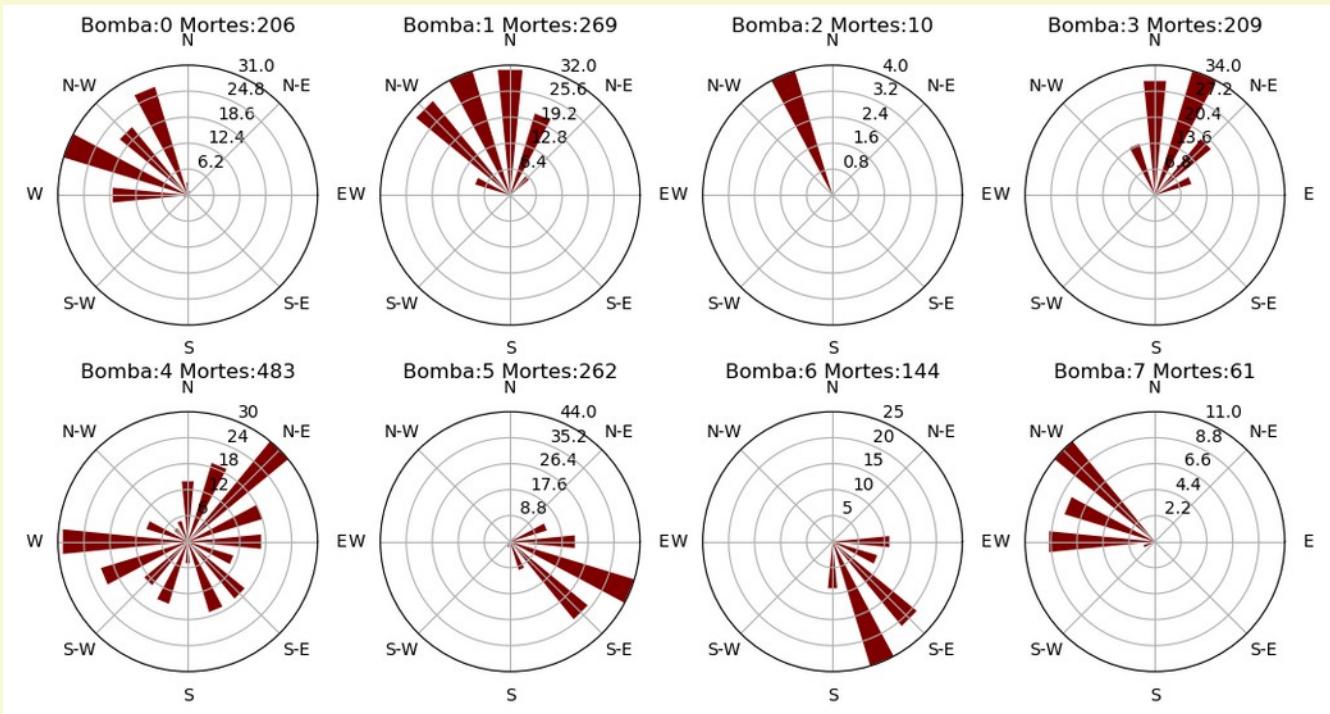
*uma matriz (array) que seleciona (SELECT) os ângulos (ST_Azimuth) em graus (degrees) entre a bomba d'água e a ocorrência morte das tabelas (FROM) **bombasdagua** e **mortescolera** onde (WHERE) fiquem dentro de um raio de 250 metros entre a bomba e a ocorrência de morte (ST_DWithin) e com o id da tabela **bombasdagua***

*extraído das tabelas (FROM) **mortescolera** e **bombadagua***

onde (WHERE) fiquem num raio de 250 metros entre a bomba e a ocorrência de morte (ST_Dwthin).

*Tudo agrupado por id da tabela **bombasdagua** (GROUP BY b.id).*

Ao executarmos o código acima vemos a quantidade e direção das mortes por cólera a uma distância de cada bomba d'água num raio de 250 metros.



Podemos alternativamente executar o script abaixo (***jsnowanalise2.py***) para repetir o que o código anterior fez e também, ao mesmo tempo, carregar o resultado com o número de mortos e direções dos óbitos e relação a cada bomba d'água como uma tabela no banco de dados ***jsnow*** chamada ***r250m***.

```
#jsnowanalise2.py
from qgis.core import *
from PyQt5.QtCore import QVariant
import psycopg2 as psy2
from windrose import WindroseAxes
from matplotlib import pyplot as plt
import matplotlib.gridspec as gridspec
import numpy as np
conn=psy2.connect(user="usr",password="sbh",host="hs",port="5432",database="jsnow")
cursor=conn.cursor()
QgsApplication.setPrefixPath("/usr", True)
qgs = QgsApplication([], False)
qgs.initQgis()
cursor.execute("CREATE TABLE r250m (bomba INT, mortos INT, oct1 INT, oct2 INT, oct3 INT, oct4 INT, oct5 INT, oct6 INT, oct7 INT, oct8 INT);")
cursor.execute("SELECT AddGeometryColumn ('public','r250m','geom',32630, 'POINT',2);")
vponto = QgsVectorLayer("Point?crs=EPSG:32630", "r250m", "memory")
dPr = vponto.dataProvider()
dPr.addAttribute([QgsField("bomba", QVariant.Int), QgsField("mortos", QVariant.Int), QgsField("oct1", QVariant.Int), QgsField("oct2", QVariant.Int), QgsField("oct3", QVariant.Int), QgsField("oct4", QVariant.Int), QgsField("oct5", QVariant.Int), QgsField("oct6", QVariant.Int), QgsField("oct7", QVariant.Int), QgsField("oct8", QVariant.Int)])
vponto.updateFields()
elem = QgsFeature()
cursor.execute("SELECT b.id,sum(a.count), ARRAY(SELECT
degrees(ST_Azimuth(st_geometryN(bombasagua.geom,1),
st_geometryN(mortescolera.geom,1))) as deg FROM bombasagua,mortescolera WHERE
bombasagua.id=b.id AND ST_Dwithin(bombasagua.geom, mortescolera.geom, 250)),
```

```

ST_X(st_geometryN(b.geom,1)), ST_Y(st_geometryN(b.geom,1)) FROM mortescolera a,
bombasdagua b WHERE ST_Dwithin(a.geom, b.geom, 250) GROUP BY b.id,b.geom;")
linhas = cursor.fetchall()
fig = plt.figure(figsize=(18, 12))
gs = gridspec.GridSpec(2, 4)
base, topo, esq, dire = gs.get_grid_positions(fig)
l=0
for linha in linhas:
    oc1,oc2,oc3,oc4,oc5,oc6,oc7,oc8 = 0,0,0,0,0,0,0,0
    bomba=linha[0]
    mortes=linha[1]
    lin, col = (l//4)%3 , l%4
    ret = [esq[col],base[lin],dire[col]-esq[col],0.9*(topo[lin]-base[lin])]
    rosa = WindroseAxes(fig, ret)
    fig.add_axes(rosa)
    angulos=linha[2]
    for angulo in angulos:
        if angulo>=0 and angulo<45: oc1=1
        if angulo>=45 and angulo<90: oc2=1
        if angulo>=90 and angulo<135: oc3=1
        if angulo>=135 and angulo<180: oc4=1
        if angulo>=180 and angulo<225: oc5=1
        if angulo>=225 and angulo<270: oc6=1
        if angulo>=270 and angulo<315: oc7=1
        if angulo>=315 and angulo<360: oc8=1
    quantidade = [1]*len(angulos)
    rosaQ = np.array(quantidade)
    rosaA = np.array(angulos)
    rosa.box(rosaA, rosaQ)
    plt.title("Bomba:"+str(bomba)+ " Mortes:"+str(mortes),pad=12.)
    elem.setGeometry(QgsGeometry.fromPointXY(QgsPointXY(linha[3],linha[4])))
    elem.setAttributes([bomba, mortes, oc1, oc2, oc3, oc4, oc5, oc6, oc7, oc8 ])
    l=l+1
    dPr.addFeature(elem)
for ponto in vponto.getFeatures():
    geom = ponto.geometry()
    cursor.execute("INSERT INTO r250m (bomba, mortos, oct1, oct2, oct3, oct4, oct5,
oct6, oct7, oct8, geom) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s,
ST_GeomFromText(%s,32630));", (ponto['bomba'],ponto['mortos'], ponto['oct1'],
ponto['oct2'], ponto['oct3'], ponto['oct4'], ponto['oct5'], ponto['oct6'],
ponto['oct7'], ponto['oct8'], geom.asWkt()))
conn.commit()
qgs.exitQgis()
plt.show()

```

Ou podemos gravar o resultado como um arquivo GIS local usando o script **jsnowanalise3.py** abaixo.

```

#jsnowanalise3.py
from qgis.core import *
from PyQt5.QtCore import QVariant
import psycopg2 as psy2
from windrose import WindroseAxes
from matplotlib import pyplot as plt
import matplotlib.gridspec as gridspec
import numpy as np
conn=psy2.connect(user="usr",password="snh",host="hs",port="5432",database="jsnow")
cursor=conn.cursor()
QgsApplication.setPrefixPath("/usr", True)
qgs = QgsApplication([], False)
qgs.initQgis()

```

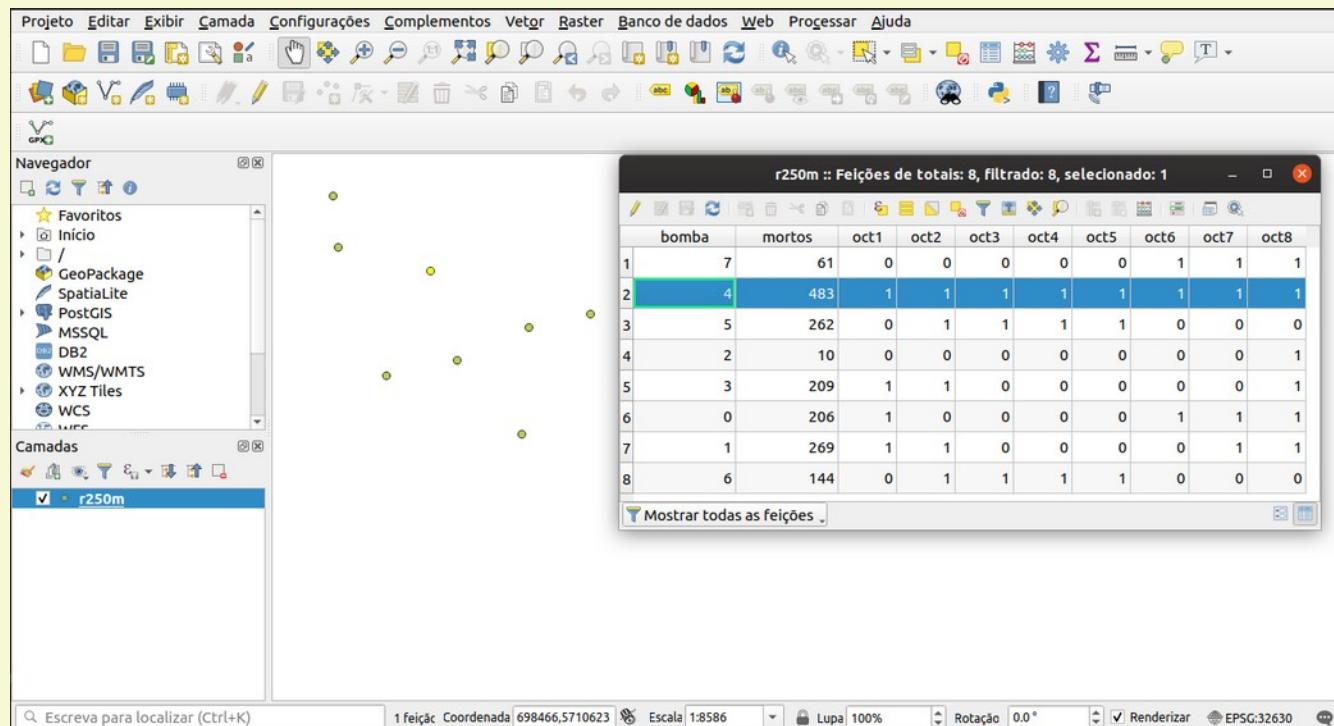
```

vponto = QgsVectorLayer("Point?crs=EPSG:32630", "r250m", "memory")
dPr = vponto.dataProvider()
dPr.addAttribute([QgsField("bomba", QVariant.Int), QgsField("mortos", QVariant.Int), QgsField("oct1", QVariant.Int), QgsField("oct2", QVariant.Int), QgsField("oct3", QVariant.Int), QgsField("oct4", QVariant.Int), QgsField("oct5", QVariant.Int), QgsField("oct6", QVariant.Int), QgsField("oct7", QVariant.Int), QgsField("oct8", QVariant.Int)])
vponto.updateFields()
elem = QgsFeature()
cursor.execute("SELECT b.id,sum(a.count),ARRAY(SELECT
degrees(ST_Azimuth(st_geometryN(bombasagua.geom,1),
st_geometryN(mortescolera.geom,1))) as deg FROM bombasagua,mortescolera WHERE
bombasagua.id=b.id AND ST_Dwithin(bombasagua.geom, mortescolera.geom, 250)),
ST_X(st_geometryN(b.geom,1)), ST_Y(st_geometryN(b.geom,1)) FROM mortescolera a,
bombasagua b WHERE ST_Dwithin(a.geom, b.geom, 250) GROUP BY b.id,b.geom;") 
linhas = cursor.fetchall()
fig = plt.figure(figsize=(18, 12))
gs = gridspec.GridSpec(2, 4)
base, topo, esq, dire = gs.get_grid_positions(fig)
l=0
for linha in linhas:
    oc1,oc2,oc3,oc4,oc5,oc6,oc7,oc8 = 0,0,0,0,0,0,0,0
    bomba=linha[0]
    mortes=linha[1]
    lin, col = (l//4)%3 , 1%4
    ret = [esq[col],base[lin],dire[col]-esq[col],0.9*(topo[lin]-base[lin])]
    rosa = WindroseAxes(fig, ret)
    fig.add_axes(rosa)
    angulos=linha[2]
    for angulo in angulos:
        if angulo>=0 and angulo<45: oc1=1
        if angulo>=45 and angulo<90: oc2=1
        if angulo>=90 and angulo<135: oc3=1
        if angulo>=135 and angulo<180: oc4=1
        if angulo>=180 and angulo<225: oc5=1
        if angulo>=225 and angulo<270: oc6=1
        if angulo>=270 and angulo<315: oc7=1
        if angulo>=315 and angulo<360: oc8=1
    quantidade = [1]*len(angulos)
    rosaQ = np.array(quantidade)
    rosaA = np.array(angulos)
    rosa.box(rosaA, rosaQ)
    plt.title("Bomba:"+str(bomba)+ " Mortes:"+str(mortes),pad=12.)
    elem.setGeometry(QgsGeometry.fromPointXY(QgsPointXY(linha[3],linha[4])))
    elem.setAttributes([bomba, mortes, oc1, oc2, oc3, oc4, oc5, oc6, oc7, oc8 ])
    l=l+1
    dPr.addFeature(elem)
QgsVectorFileWriter.writeAsVectorFormat(vponto, '/home/usr/r250m.shp', 'utf-8',
driverName='ESRI Shapefile')
conn.commit()
qgs.exitQgis()
plt.show()

```

O resultado da análise

A imagem gerada acima já deixa bem claro qual é o mais forte candidato a ser a bomba d'água contaminada. Vamos abrir a tabela do banco de dados ou o arquivo resultante e identificar qual delas é a fonte contaminadora. Vale lembrar que naquela época não se sabia que a cólera era transmitida por fonte de água contaminada e essa foi a grande descoberta do Dr. John Snow com sua análise espacial. A bomba d'água com o maior número de mortes num raio de 250 metros e com mortos em todas as direções foi a de **número 4**.



A solução foi remover a torneira que abria a fonte 4 e o surto de cólera em Soho foi eliminado, graças ao resultado alcançado pela análise geoespacial.

6.1.2 Exemplo 1 – Um outro exemplo usando somente pyQgis

Um grande amigo de seu Melbourne te convida para uma festa de despedida com 100 convidados. Como ele partirá para uma missão científica de 4 anos você ficou encarregado de escolher qual o bar será a festa. Como você não conhece sobre lá mas conhece os hábitos alcoólicos de seus amigos você precisa de encontrar um bar que seja próximo de uma estação de metro para garantir o bom retorno de todos após a festa. Também não pode ser um local muito grande, 200 pessoas no máximo.

Preparando

No site da prefeitura de Melbourne você conseguiu uma lista dos bares com capacidade (**bares.csv**) e também o shapefile das estações de metrô (**estacoes_metro.***). Vamos carregar estas informações baixando estes dois arquivos localmente em seu sistema. Este dado será usado para cruzarmos as informações e validar os critérios necessários.

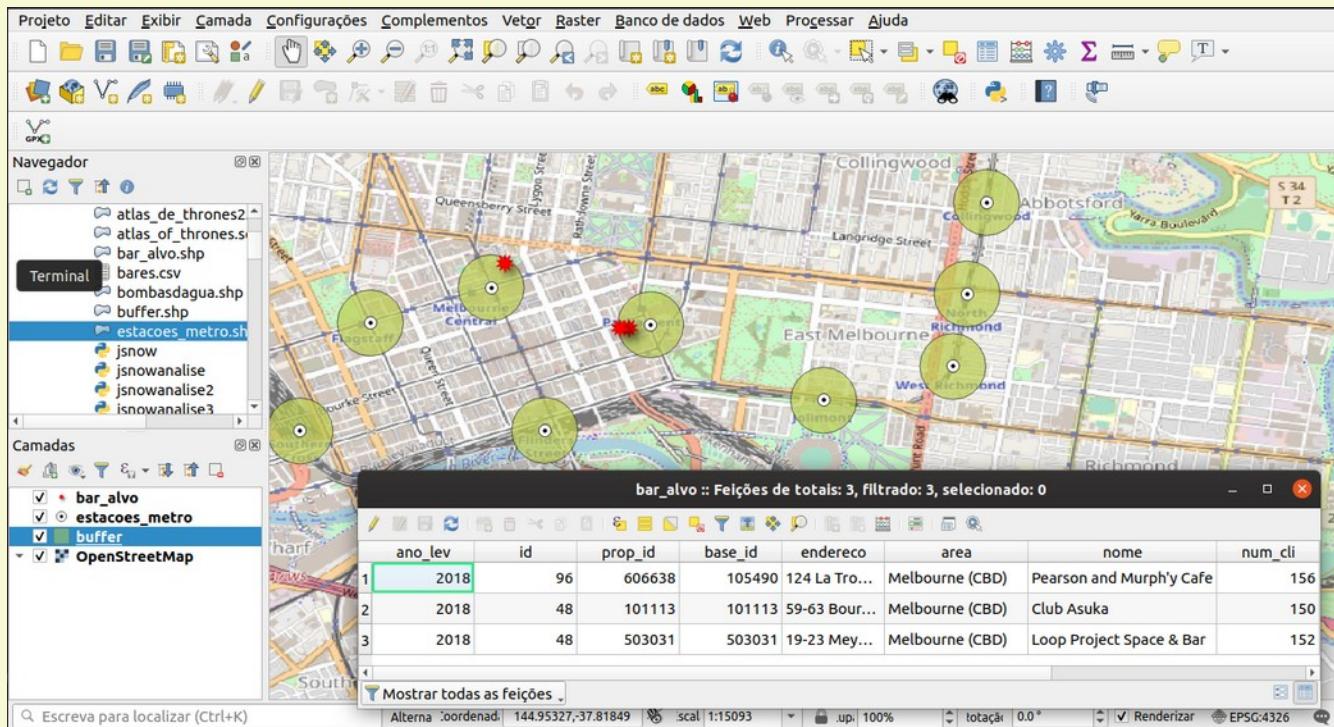
Executando

Executaremos a análise espacial com o script **melbar.py** abaixo usando os critérios que precisamos.

```
# melbar.py
# importando a library e iniciando o qgis
from qgis.core import *
QgsApplication.setPrefixPath("/usr", True)
qgs = QgsApplication([], False)
qgs.initQgis()
# Carregando os dados no como objetos vetoriais
uri="file:///home/usr/bares.csv?type=csv&xField=x&yField=y&crs=epsg:4326"
bares = QgsVectorLayer(uri, 'bares', "delimitedtext")
estacoes = QgsVectorLayer("/home/usr/estacoes_metro.shp","estacoes", "ogr")
# definindo os arquivos de saida da análise
arqBuf = "/home/usr/buffer.shp" # arquivo buffer de saída
baralvo = "/home/usr/bar_alvo.shp" # arquivo com resultado da análise
# variáveis de parâmetros e campos com os atributos
bufDist = 0.002 #aproximadamente 200 metros de raio
fields = estacoes.fields()
bfields = bares.fields()
feats = estacoes.getFeatures()
# criando o objeto que escreve o arquivo de buffer
writer=QgsVectorFileWriter(arqBuf,'UTF-8',fields, QgsWkbTypes.Polygon,
estacoes.sourceCrs(), 'ESRI Shapefile')
# Criando polígono de buffer de ~ 200 m em cada estação do metrô
for feat in feats:
    geom = feat.geometry()
    buff = geom.buffer(bufDist,5)
    feat.setGeometry(buff)
    writer.addFeature(feat)
del(writer)
# criando o objeyo que escreve o arquivo com o resultado da análise
writer=QgsVectorFileWriter(baralvo,'UTF-8',bfields, QgsWkbTypes.Point,
bares.sourceCrs(), 'ESRI Shapefile')
# lendo o arquivo de buffer
bufa = QgsVectorLayer("/home/usr/buffer.shp","bufa", "ogr")
# carregando os dados do buffer e dos bares
bufaFeatureList = [plFeat for plFeat in bufa.getFeatures()]
baresFeatureList = [ptFeat for ptFeat in bares.getFeatures()]
# criando o arquivo de bares que atendem aos critérios da análise
for ptFeat in baresFeatureList:
    ptGeom = ptFeat.geometry()
    for plFeat in bufaFeatureList:
        plGeom = plFeat.geometry()
        if plGeom.contains(ptGeom) and ptFeat[7]>100 and ptFeat[7]<200 and
ptFeat[0]==2018: # checa nos dados de 2018 se está dentro do buffer, e entre 100 e
200 pessoas em capacidade
            writer.addFeature(ptFeat)
del(writer)
# finaliza
qgs.exitQgis()
```

O resultado da análise

O resultado atingido ao abrirmos o arquivo bar_alvo.shp no Qgis apresentou 3 opções de bares que poderemos fazer a festa de despedido do Nick:



6.1.3 Considerações sobre as duas análises espaciais básicas efetuadas

Mostramos como análises básicas podem levar aos resultados desejados em poucos passos. O objetivo principal aqui foi mostrar que podemos usar tanto o Postgis via query quanto o pyQgis via classes e métodos para filtrar e/ou avaliar os dados espaciais.

As análises que fizemos, embora eficientes, foram bem simples quando comparado ao potencial tanto do Postgis e do pyQgis. Vamos nas sessões seguintes ver como podemos executar análises espaciais complexas usando outras libraries (**geopandas** e **rasterio**) em conjunto com pyQgis.

Na próxima sessão vamos cobrir análise de dados que apresentam como resultado objetos vetoriais e na seguinte análise de dados que resultam em raster.

7. Análise espacial em dados vetoriais

A análise espacial com dados vetoriais é o cerne da lógica que relaciona e cruza as informações com a distribuição espacial. Para tornar esse processo eficiente se faz necessário o uso de libraries focadas na melhor estruturação e acesso a essas informações. Geopandas, além de compacto, é uma maneira fácil e eficiente de organizar e intermediar o processo da análise espacial. Vamos cobrir aqui o uso dele e avançar o processo de análise espacial.

7.1 Geopandas crash course

7.1.1 Instalação

Instale o **geopandas** usando conda ou pip3, para plotarmos os objetos espaciais também instale o **descartes** usando conda ou pip3. Se for trabalhar com geocoding, instale o **geopy** também.

7.1.2 O básico

Vamos usar o arquivo shapefile (TO.*) para demonstrar os aspectos básicos do geopandas. Vamos cobrir aqui os comandos mais usados e o que fazem objetivamente, para maiores detalhes sobre cada classe e método use a documentação em <http://geopandas.org/index.html>.

Entrando com os dados

Importe as libraries e carregue o arquivo usando **gpd.read(arquivo)**. Onde *arquivo* pode ser qualquer tipo de arquivo gis suportado pela sua instalação de GDAL

```
>>> import geopandas as gpd  
>>> to = gpd.read_file('TO.shp')  
>>> type(to)  
<class 'geopandas.geodataframe.GeoDataFrame'>
```

Obtendo informações do objeto

Extraindo a informação da extensão geográfica (geometrica) total dos dados com **total_bounds**.

```
>>> to.total_bounds  
array([-50.66151341, -13.50015585, -46.04695556, -5.15689212])
```

Extraindo a informação da extensão dos 4 primeiros elementos com **bounds**.

```
>>> to[0:4].bounds  
   minx      miny      maxx      maxy  
0 -47.584763 -7.391760 -47.562118 -7.373675  
1 -49.644115 -10.849308 -49.609820 -10.813146  
2 -48.279049 -13.074260 -48.269829 -12.983869  
3 -48.306697 -13.074279 -48.297476 -12.983888
```

Obtendo o nome de cada coluna do objeto com **columns**.

```
>>> to.columns  
Index(['PROCESSO', 'ID', 'NUMERO', 'ANO', 'AREA_HA', 'FASE', 'ULT_EVENTO',  
       'NOME', 'SUBS', 'USO', 'UF', 'geometry'],  
      dtype='object')
```

E o tipo de dado de cada coluna (tipo texto aparece como tipo objeto em geopandas) com dtypes.

```
>>> to.dtypes
PROCESSO      object
ID            object
NUMERO        int64
ANO            int64
AREA_HA       float64
FASE          object
ULT_EVENTO    object
NOME          object
SUBS          object
USO            object
UF             object
geometry      geometry
dtype: object
```

O sistema de referência das coordenadas são obtidos com crs

```
>>> to.crs
{'init': 'epsg:4674'}
```

Dados do Postgis

Obtemos um objeto diretamente do postgis para o geopandas usando.

```
>>> import psycopg2 as psy
>>> con = psy.connect(database="dnpmto", user="droid", password="devcor",
host="amazeone.com.br")
>>> sql = "select * from gis"
>>> to2 = gpd.GeoDataFrame.from_postgis(sql, con, geom_col='geom' )
```

Filtrando dados (query)

Podemos filtrar de forma simples os dados de um objeto usando query.

```
>>> conLav=to.query('FASE == "CONCESSÃO DE LAVRA" ')
```

Gravando objeto em arquivo

Gravar um objeto geopandas em arquivo gis é tão simples como ler o objeto do arquivo, basta definir o caminho com o nome do arquivo e usar o método **to_file(arquivo)**.

```
>>> arquiGrava = "/home/usr/TOconLav.shp"
>>> conLav.to_file(arquiGrava)
```

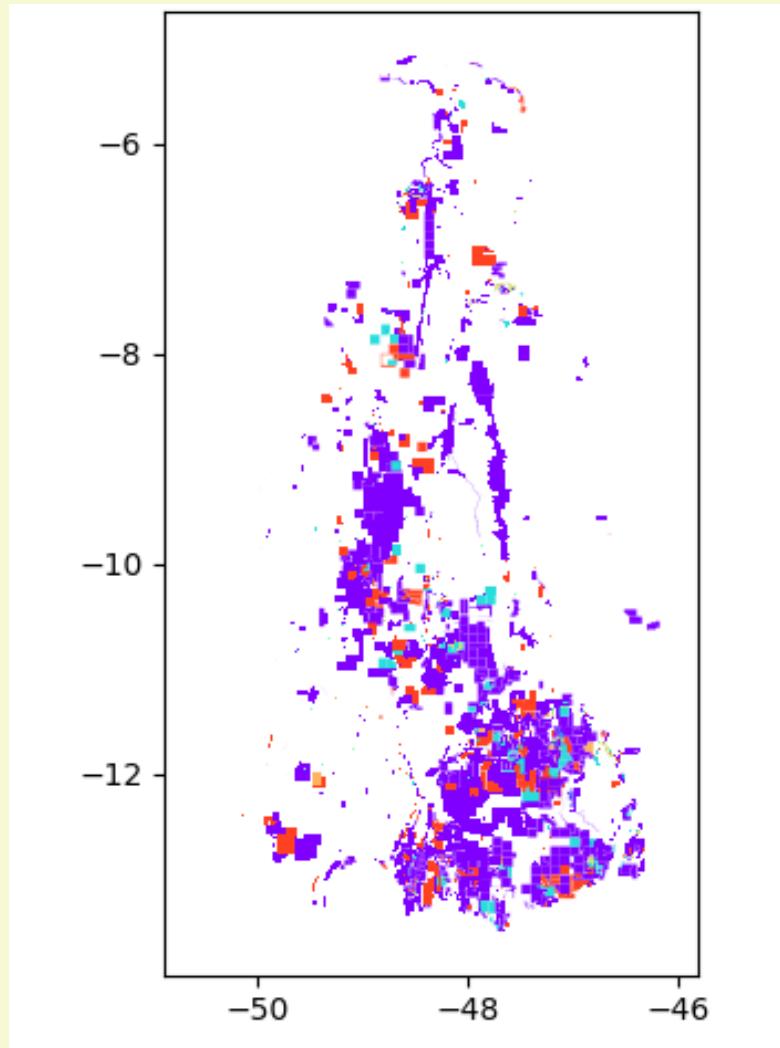
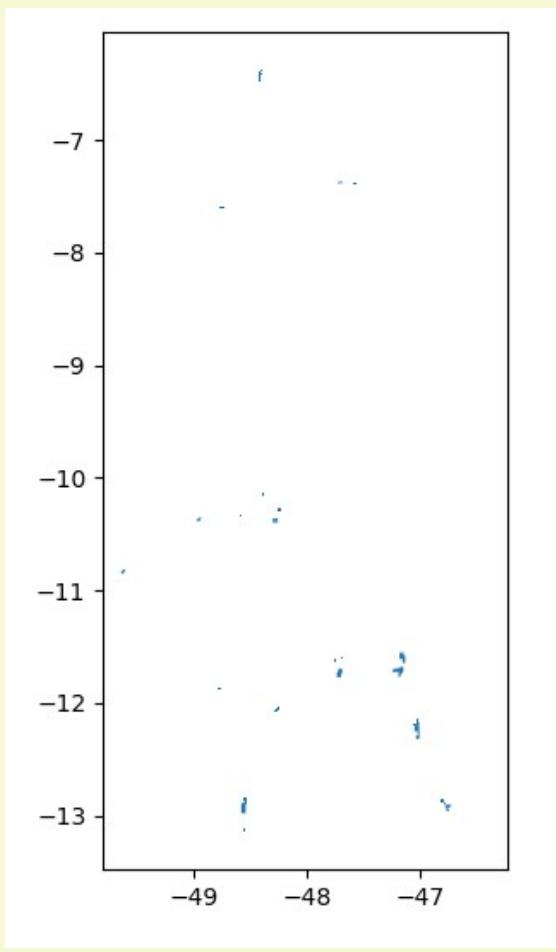
Plotando o objeto

Usando pyplot em conjunto com geopandas podemos plotar um objeto para rápida visualização usando o método **plot()**.

```
>>> from matplotlib import pyplot as plt
>>> conLav.plot()
>>> plt.show()
```

E até mesmo fazer uma rápida classificação por uma das colunas de atributos do objeto.

```
>>> to.plot(column='FASE', cmap='rainbow');
>>> plt.show()
```



7.1.3 Relações espaciais de geometrias com o Geopandas

O lado forte de geopandas é a forma simples de fazer análise espacial entre dois objetos espaciais, vamos agora ver alguns deles.

Checando a posição relativa

As seguintes funções resultam em uma serie (coluna) com os valores True ou False de acordo com a interação do objeto que chama com o objeto comparado.

contains(outro objeto)

Retorna uma coluna do tipo 'bool' com o valor True se cada objeto interior contém integralmente o objeto pasado no argumento (as bordas não se tocam).

crosses(outro objeto)

Retorna uma coluna do tipo 'bool' com o valor True se o interior do objeto intercepta o interior do outro objeto mas não o contém inteiramente e a dimensão da intersecção é menor que a dimensão de qualquer um dos dois objetos.

disjoint(outra objeto)

Retorna uma coluna do tipo 'bool' com o valor True se a borda e o interior de cada objeto não intersectam de forma alguma com o outro objeto.

equals(outra objeto)

Retorna uma coluna do tipo 'bool' com o valor True se a borda, o interior e o exterior coincidem exatamente com aquele do outro objeto.

intersects(outra objeto)

Retorna uma coluna do tipo 'bool' com o valor True se a borda e o interior de cada objeto intersectam de alguma forma o interior e a borda do outro objeto.

touches(outra objeto)

Retorna uma coluna do tipo 'bool' com o valor True se o objeto tem pelo menos um ponto em comum com o outro objeto e seus interiores não se intersectam.

within(outra objeto)

Retorna uma coluna do tipo 'bool' com o valor True se cada borda do objeto e seu interior interceptam somente com o interior do outro objeto.

Retornando uma nova geometria

Estes métodos retornam um novo objeto resultante do que é passado como parâmetro, de como um objeto interage com ele mesmo ou com outro objeto.

unary_union

Retorna uma geometria contendo a união de todas as geometrias no objeto GeoSeries.

boundary

Retorna uma GeoSeries de objetos de dimensão menor representando cada conjunto de geometrias como uma borda teórica.

centroid

Retorna uma GeoSeries de pontos para cada centro de geometria do objeto.

difference(outra objeto)

Retorna uma GeoSeries de pontos para cada geometria que não estão presentes no outro objeto.

intersection(outra objeto)

Retorna uma GeoSeries da intersecção de cada objeto com o outro objeto geométrico. (onde esses se sobrepõem).

GeoSeries.symmetric_difference(outra objeto)

Retorna uma GeoSeries de pontos em cada objeto que não está no outro objeto e também pontos em cada objeto do outro objeto que não está nesse objeto.

union(outra objeto)

Retorna uma GeoSeries da união dos pontos de cada objeto e os outros objetos geométricos.

Métodos construtivos

Estes métodos constroem novos objetos com base no conjunto de dados de todos os objetos ou de cada um individualmente.

buffer(distância, resolution=16)

Retorna uma GeoSeries de geometrias representando todos os pontos dentro de uma distância dada de cada objeto geométrico.

convex_hull

Retorna uma GeoSeries de geometrias representando o menor polígono convexo que envolve todos os pontos de cada objeto. Caso o número de pontos no objeto seja pelo três. Para um objeto com dois pontos convex_hull transforma o objeto em uma LineString; Se tiver 1, um Ponto.

envelope

Retorna uma coluna de GeoSeries de geometrias representando o ponto ou o menor polígono retangular ortogonal que contenha cada objeto.

simplify(tolerância, preserve_topology=True)

Retorna uma GeoSeries contendo uma representação simplificada de cada objeto.

7.1.4 Como ficariam os dois exemplos da sessão 1 usando Geopandas

Vamos agora mostrar como usar o geopandas nos dois exemplos que fizemos na sessão anterior onde analisamos os casos de cólera e a seleção do bar em Melbourne

Casos de cólera

No script **jsnowpanda.py** abaixo está é versão da mesma análise efetuada antes, somente que usamos geopandas para gerar o arquivo shapefile **r250mGPD.shp** final.

```
#jsnowpanda.py
import geopandas as gpd
import psycopg2 as psy2
from windrose import WindroseAxes
from matplotlib import pyplot as plt
import matplotlib.gridspec as gridspec
import numpy as np
conn=psy2.connect(user="usr",password="snh",host="hs",port="5432",database="jsnow")
sql = "SELECT b.id as bomba,sum(a.count) as mortes,ARRAY(SELECT
degrees(ST_Azimuth(st_geometryN(bombasdagua.geom,1),
st_geometryN(mortescolera.geom,1))) as deg FROM bombasdagua,mortescolera WHERE
bombasdagua.id=b.id AND ST_Dwithin(bombasdagua.geom, mortescolera.geom, 250)),
ST_GeometryN(b.geom,1) as geom FROM mortescolera a, bombasdagua b WHERE
ST_Dwithin(a.geom, b.geom, 250) GROUP BY b.id,b.geom;""
dados = gpd.GeoDataFrame.from_postgis(sql, conn, geom_col='geom' )
dados=dados.assign(oc1=0,oc2=0,oc3=0,oc4=0,oc5=0,oc6=0,oc7=0,oc8=0)
nco=int(dados.size/dados.columns.size)
fig = plt.figure(figsize=(18, 12))
gs = gridspec.GridSpec(2, 4)
base, topo, esq, dire = gs.get_grid_positions(fig)
l=0
```

```

for linha in range(0,nco):
    angulos=dados['array'][linha]
    oc1,oc2,oc3,oc4,oc5,oc6,oc7,oc8 = 0,0,0,0,0,0,0,0
    lin, col = (l//4)%3 , l%4
    ret = [esq[col],base[lin],dire[col]-esq[col],0.9*(topo[lin]-base[lin])]
    rosa = WindroseAxes(fig, ret)
    fig.add_axes(rosa)
    for angulo in angulos:
        if angulo>=0 and angulo<45: oc1=1
        if angulo>=45 and angulo<90: oc2=1
        if angulo>=90 and angulo<135: oc3=1
        if angulo>=135 and angulo<180: oc4=1
        if angulo>=180 and angulo<225: oc5=1
        if angulo>=225 and angulo<270: oc6=1
        if angulo>=270 and angulo<315: oc7=1
        if angulo>=315 and angulo<360: oc8=1
    quantidade = [1]*len(angulos)
    rosaQ = np.array(quantidade)
    rosaA = np.array(angulos)
    rosa.box(rosaA, rosaQ)
    plt.title("Bomba:"+str(dados['bomba'][linha])+" Mortes:"+str(dados['mortes'][linha]),pad=12.)
    dados['oc1'][linha],dados['oc2'][linha],dados['oc3'][linha],dados['oc4'][linha],dados['oc5'][linha],dados['oc6'][linha],dados['oc7'][linha],dados['oc8'][linha] =oc1,oc2,oc3,oc4,oc5,oc6,oc7,oc8
    l=l+1
del dados['array']
dados.to_file('/home/usr/r250mGPD.shp')
plt.show()

```

Festa em Melbourne

Está é versão da mesma análise efetuada antes para encontrar o bar em Melbourne, somente que usamos geopandas para gerar o arquivo shapefile **bar_alvoGPD.shp** final.

#melbarGPD.py

```

import geopandas as gpd
import pandas as pd
from shapely.geometry import Point
bares=pd.read_csv('bares.csv')
bares['geometry'] = bares.apply(lambda row: Point(row.x, row.y), axis=1)
bares = gpd.GeoDataFrame('/home/usr/bares,crs='epsg:4326')
bares = bares.query('ano_lev == 2018')
bares = bares.query('num_cli > 100 and num_cli < 200')
metro=gpd.read_file('/home/usr/estacoes_metro.shp')
noRaio = metro['geometry'].buffer(0.002)
bor = bares.within(noRaio.unary_union)
bares = bares.loc[bor]
bares.to_file('/home/usr/bar_alvoGPD.shp')

```

7.2 Case study – Análise espacial com dados de exploração mineral no Tocantins

Vamos fazer uma análise espacial usando geopandas com o objetivo de fixar o uso das relações espaciais de objetos. Vamos portanto, de forma preliminar, selecionar áreas potenciais para ocorrência de jazidas de minerais metálicos.

7.2.1 Os dados vetoriais que serão usados

A primeira etapa no processo de análise é obter os dados e conhecer a sua estrutura.

Carregando os dados

Os dados, para essa análise, estão disponíveis no banco de dados **tocantins** no host **amazeone.com.br** e o usuário de acesso é **droid** com senha **devcor**.

Acesse o banco de dados, abra todas as camadas e grave localmente como arquivo shapefile as camadas geologia, recmin e requerimentos.

Conhecendo os dados

Vamos acionar o python e examinar a estrutura de cada dado. Primeiramente listando as colunas de cada objeto.

```
>>> import geopandas as gpd
>>> recmin=gpd.read_file('recmin.shp',encoding='utf-8')
>>> geologia=gpd.read_file('geologia.shp',encoding='utf-8')
>>> req=gpd.read_file('requerimentos.shp',encoding='utf-8')
>>> recmin.columns
Index(['CODIGO_OBJ', 'LATITUDE', 'LONGITUDE', 'DATUM', 'DATA_CADAS',
       'SUBST_PRIN', 'SUBST_SECU', 'ABREVIACAO', 'ASSOC_MINE', 'ASSOCIACAO',
       'CLASSE_GEN', 'CLASSE_UTI', 'EXTRMIN_X', 'GRAU_DE_IM', 'METODO_GEO',
       'ERRO_METOD', 'ROCHA_ENCA', 'ROCHA_HOSP', 'TEXTURA_MI', 'MODELO_DEP',
       'TIPOLOGIA', 'ORIGEM', 'TOponimia', 'STATUS_ECO', 'TIPOS_ALTE',
       'MUNICIPIO', 'UF', 'RESE_MEDI', 'CRIT_MEDI', 'BIBL_MEDI', 'RESE_INDI',
       'CRIT_INDI', 'BIBL_INDI', 'RESE_INFE', 'CRIT_INFE', 'BIBL_INFE',
       'FONTE', 'OBS', 'geometry'],
      dtype='object')
>>> geol.columns
Index(['SIGLA_UNID', 'NOME_UNIDA', 'HIERARQUIA', 'LITOTIPO1', 'LITOTIPO2',
       'CLASSE_RX1', 'CLASSE_RX2', 'COD_DOM', 'DOM_GEO', 'COD_UNIGEO',
       'UNIGEO', 'DEF_TEC', 'CIS_FRAT', 'TIPO_DEF', 'COM_REOL', 'ASPECTO',
       'INTEM_FIS', 'INTEM QUI', 'GR_COER', 'TEXTURA', 'PORO_PRI',
       'LITO_HIDRO', 'COD_REL', 'RELEVO', 'DECLIVIDAD', 'AMPLITUDE', 'GEO_Reo',
       'geometry'],
      dtype='object')
>>> req['FASE'].unique()
array(['CONCESSÃO DE LAVRA', 'DIREITO DE REQUERER A LAVRA',
       'AUTORIZAÇÃO DE PESQUISA', 'REQUERIMENTO DE LAVRA',
       'LICENCIAMENTO', 'REQUERIMENTO DE PESQUISA', 'LAVRA GARIMPEIRA',
       'DISPONIBILIDADE', 'REQUERIMENTO DE LICENCIAMENTO',
       'REQUERIMENTO DE LAVRA GARIMPEIRA',
       'REQUERIMENTO DE REGISTRO DE EXTRAÇÃO', 'REGISTRO DE EXTRAÇÃO',
       'DADO NÃO CADASTRADO'], dtype=object)
```

Vamos agora ver as opções de valores presentes nas colunas que usaremos como filtro na próxima etapa de nossa análise espacial.

```
>>> recmin['ABREVIACAO'].unique()
array(['Au', 'Gr', 'Cc', 'Ccd', 'Gp', 'Are, Sx', 'Arg', 'Are, Sa',
       'Are, Cas', 'Agm', 'Sn', 'Qz', 'Qzh', 'Di', 'Di, Au', 'Au, Di',
       'Fp', 'Pb, S, Zn, Ag, Cu, Cd, Au', 'Cu, Pb, Cd, Zn, Ag',
       'Cu, Pb, Cd, Au, Ag, Zn', 'Zn, Pb, Cd, Au, Cu, Ag', 'Cr', 'Tal',
       'Cli', 'P', 'Tu', 'Ti', 'Cu', 'Zr', 'Mn', 'Sm'], dtype=object)
>>> req['FASE'].unique()
array(['CONCESSÃO DE LAVRA', 'DIREITO DE REQUERER A LAVRA',
       'AUTORIZAÇÃO DE PESQUISA', 'REQUERIMENTO DE LAVRA',
       'LICENCIAMENTO', 'REQUERIMENTO DE PESQUISA', 'LAVRA GARIMPEIRA',
       'DISPONIBILIDADE', 'REQUERIMENTO DE LICENCIAMENTO',
       'REQUERIMENTO DE LAVRA GARIMPEIRA',
       'REQUERIMENTO DE REGISTRO DE EXTRAÇÃO', 'REGISTRO DE EXTRAÇÃO',
       'DADO NÃO CADASTRADO'], dtype=object)
>>> req['SUBS'].unique()
array(['GIPSITA', 'DOLOMITO', 'NÍQUEL', 'CALCÁRIO', 'WOLFRAMITA', 'PRATA',
       'CROMO', 'OURO', 'CALCÁRIO CALCÍTICO', 'CALCÁRIO DOLOMÍTICO',
       'ZINCO', 'COBRE', 'MINÉRIO DE OURO', 'SAIS DE POTÁSSIO', 'TÂNTALO',
       'GRANITO', 'TITÂNIO', 'ESTANHO', 'GNAISSE', 'SÍLEX', 'FELDSPATO',
       'ÁGUA MINERAL', 'ARGILA', 'MINÉRIO DE ZIRCÔNIO', 'AREIA',
       'QUARTZO', 'ZIRCONITA', 'QUARTZITO', 'MINÉRIO DE NÍQUEL',
       'DIAMANTE', 'ZIRCÔNIO', 'AREIA QUARTZOSA', 'TERRAS RARAS',
       'MANGANÊS', 'MINÉRIO DE FERRO', 'BASALTO', 'MINÉRIO DE MANGANÊS',
       'FOSFATO', 'ESMERALDA', 'ILMENITA', 'MINÉRIO DE TÂNTALO',
       'CASCALHO', 'MINÉRIO DE COBRE', 'RUTILO', 'ARDÓSIA',
       'SERPENTINITO', 'CÁDMIO', 'BERILO', 'MINÉRIO DE TITÂNIO',
       'GRANADA', 'MINÉRIO DE NIÓBIO', 'MINÉRIO DE PLATINA', 'FERRO',
       'TURFA', 'MINÉRIO DE ZINCO', 'MINÉRIO DE ALUMÍNIO',
       'MINÉRIO DE CHUMBO', 'TURMALINA', 'CAULIM', 'ARGILITO', 'SIENITO',
       'MINÉRIO DE ESTANHO', 'DIABÁSIO', 'ARENITO', 'CASSITERITA',
       'TANTALITA', 'MÁRMORE', 'SAIBRO', 'GRAFITA', 'ZIRCÃO',
       'DADO NÃO CADASTRADO'], dtype=object)
```

7.2.2 A preparação dos dados

A segunda fase do processo é a preparação dos dados que serão usados na análise. A preparação pode consistir em filtragem, comparação e/ou edição de estrutura dos dados. No nosso caso vamos nos ater somente à filtragem dos dados a serem usados.

Filtrando os dados

Do objeto recmin vamos extrair somente as ocorrências minerais dos metais na coluna ABREVIACAO: 'Au'; 'Sn'; 'Pb, S, Zn, Ag, Cu, Cd, Au'; 'Cu, Pb, Cd, Zn, Ag'; 'Cu, Pb, Cd, Au, Ag, Zn'; 'Zn, Pb, Cd, Au, Cu, Ag'; 'Cr'; 'Ti'; 'Cu'; 'Zr' e 'Mn'. Criaremos um novo objeto com o resultado.

Do objeto req vamos filtrar somente as fases: 'REQUERIMENTO DE LAVRA' e 'CONCESSÃO DE LAVRA' dos seguintes metais: 'OURO', 'MINÉRIO DE OURO', 'WOLFRAMITA' e 'CROMO'. Criaremos um novo objeto com o resultado.

O script **filtragem.py** abaixo executará essas tarefas.

```
#filtragem.py
import geopandas as gpd
recmin=gpd.read_file('/home/usr/recmin.shp',encoding='utf-8')
req=gpd.read_file('/home/usr/requerimentos.shp',encoding='utf-8')
```

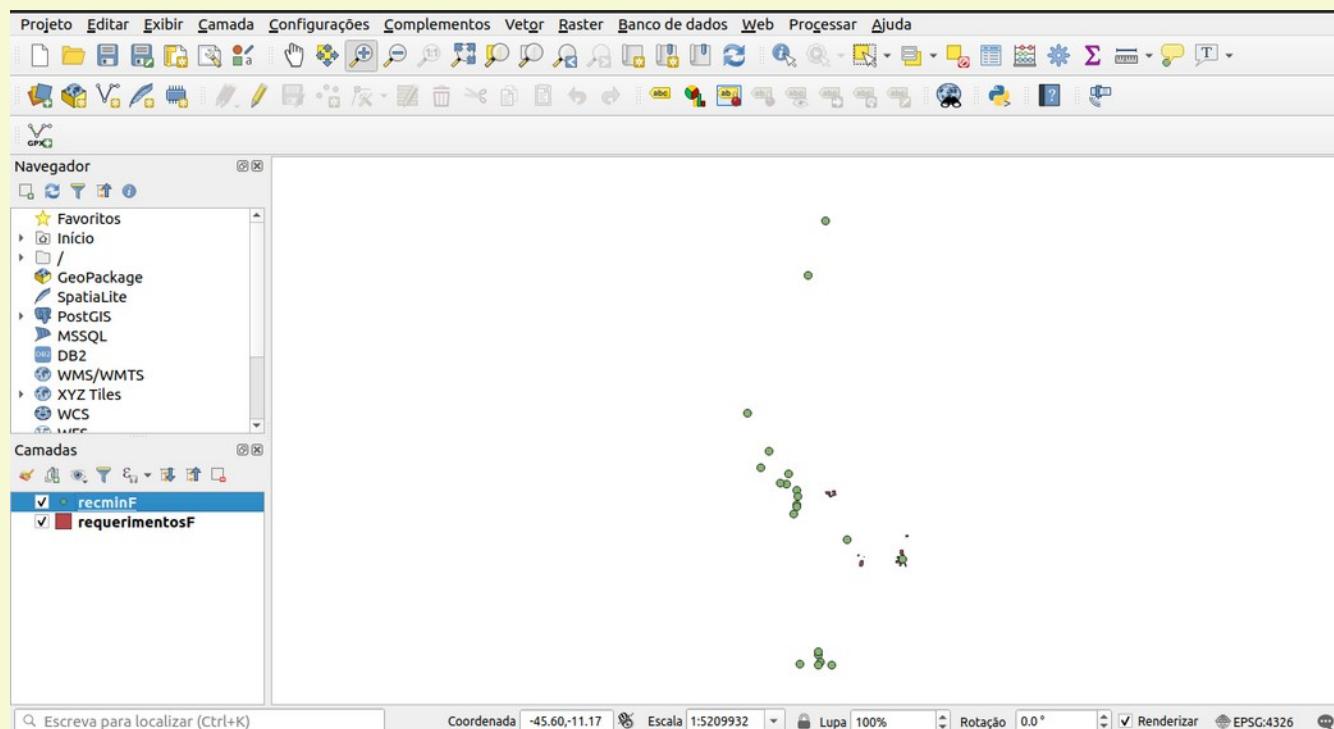
```

recmin2 = recmin.query('ABREVIACAO == "Au" or ABREVIACAO == "Sn" or ABREVIACAO == "Pb, S, Zn, Ag, Cu, Cd, Au" or ABREVIACAO == "Cu, Pb, Cd, Zn, Ag" or ABREVIACAO == "Cu, Pb, Cd, Au, Ag, Zn" or ABREVIACAO == "Zn, Pb, Cd, Au, Cu, Ag" or ABREVIACAO == "Cr" or ABREVIACAO == "Ti" or ABREVIACAO == "Cu" or ABREVIACAO == "Zr" or ABREVIACAO == "Mn" ')
req2 = req.query('FASE == "REQUERIMENTO DE LAVRA" or FASE == "CONCESSÃO DE LAVRA"')
req3 = req2.query('SUBS == "OURO" or SUBS == "MINÉRIO DE OURO" or SUBS == "WOLFRAMITA" or SUBS == "CROMO" ')
recmin2.to_file('/home/usr/recminF.shp')
req3.to_file('/home/usr/requerimentosF.shp')

```

Visualizando os dados filtrados

Abaixo temos os dois arquivos no Qgis.



Com a criação dos novos objetos filtrados podemos passar agora para a próxima etapa

7.2.3 A análise dos dados

Nesta terceira etapa criamos o mapa preliminar das unidades geológicas com presença de minerais metálicos.

Por dados de ocorrência mineral

Criamos um mapa de unidades geológicas com ocorrência de minerais metálico com o script **lista1mm.py** abaixo.

```

#lista1mm.py
import geopandas as gpd
recminF=gpd.read_file('/home/usr/recminF.shp',encoding='utf-8')
geol=gpd.read_file('/home/usr/geologia.shp',encoding='utf-8')
bor = geol.intersects(recminF.unary_union)

```

```
geolF = geol.loc[bor]
geolF.to_file('/home/usr/geologiaF1.shp')
```

Por requerimento e concessão de lavra

Criamos um mapa de unidades geológicas com requerimento de lavra ou concessão de lavra para minerais metálico com o script **lista2mm.py** abaixo.

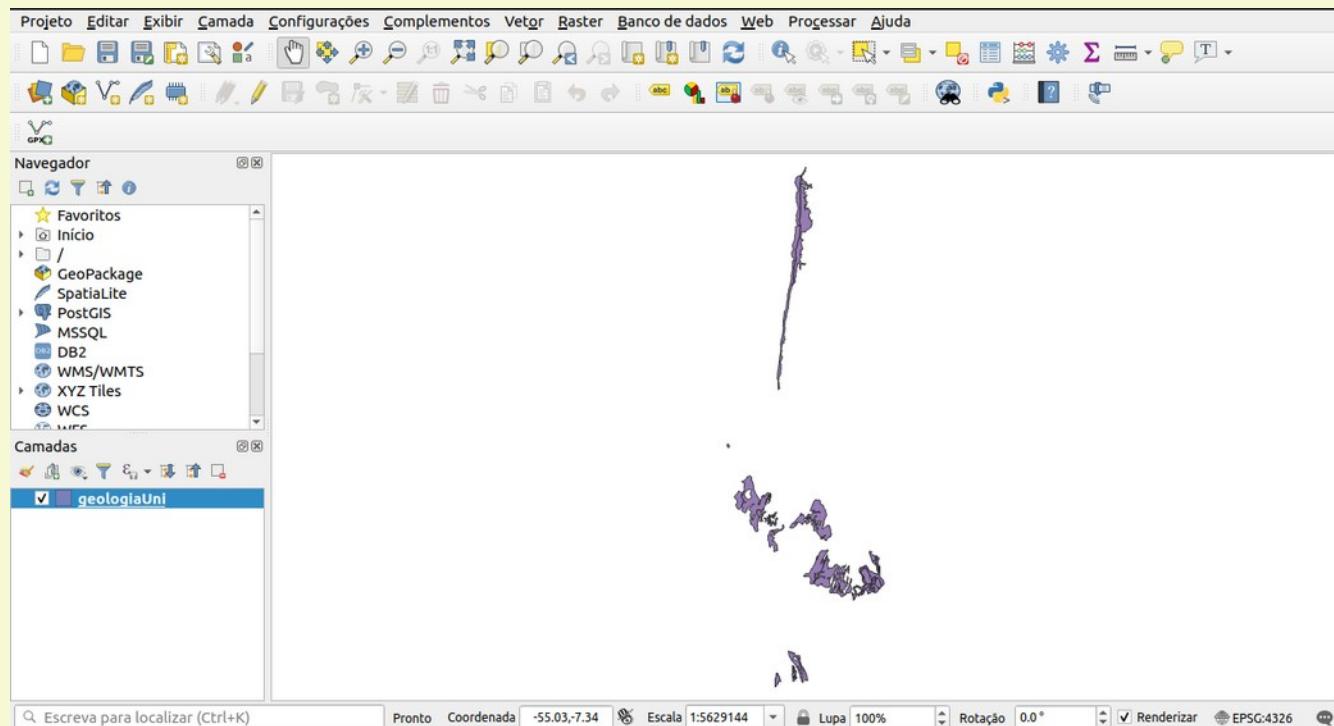
```
#lista2mm.py
import geopandas as gpd
reqF=gpd.read_file('/home/usr/requerimentosF.shp',encoding='utf-8')
geol=gpd.read_file('/home/usr/geologia.shp',encoding='utf-8')
bor = geol.intersects(reqF.unary_union)
geolF = geol.loc[bor]
geolF.to_file('/home/usr/geologiaF2.shp')
```

Unindo os resultados

Unimos os dois mapas acima com o script **mapa1mm.py** abaixo.

```
#mapa1mm.py
import geopandas as gpd
geol1=gpd.read_file('/home/usr/geologiaF1.shp',encoding='utf-8')
geol2=gpd.read_file('/home/usr/geologiaF2.shp',encoding='utf-8')
geolF = geol1.union(geol2)
geolF.to_file('/home/usr/geologiaUni.shp')
```

O mapa resultante é mostrado abaixo.



Expandindo o mapa para as unidades geológicas iguais

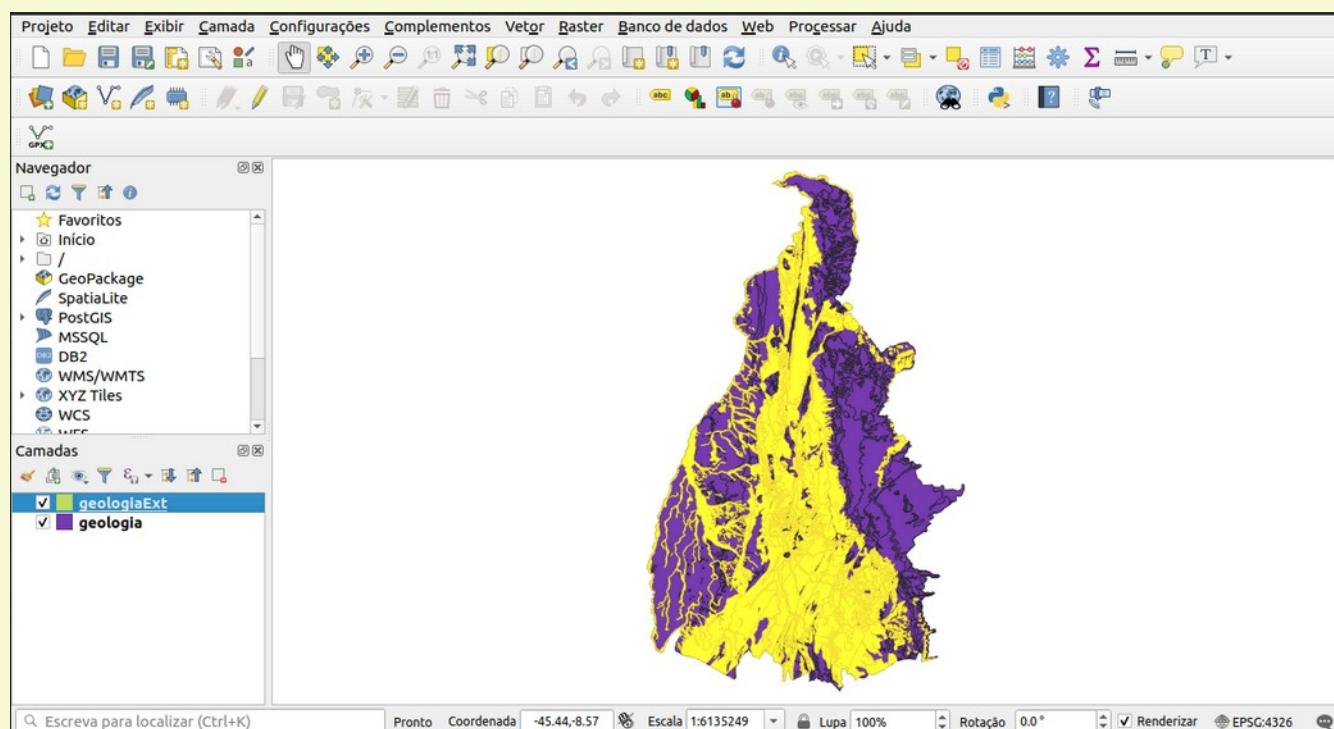
Vamos agora expandir o nosso mapa incluindo todas as unidades geológicas com as mesmas características (mesma Formação ou Grupo) das rochas que apresentaram mineralização. O script **mapa2mm.py** abaixo faz esta expansão.

```

#mapa2mm.py
import geopandas as gpd
import pandas as pd
geol=gpd.read_file('/home/usr/geologia.shp',encoding='utf-8')
geol1=gpd.read_file('/home/usr/geologiaF1.shp',encoding='utf-8')
geol2=gpd.read_file('/home/usr/geologiaF2.shp',encoding='utf-8')
geolF = pd.concat([geol1, geol2], ignore_index=True)
unidades=geolF['SIGLA_UNID'].unique()
i=0
que=''
for unidade in unidades:
    if i ==0: que='SIGLA_UNID == "'+unidade+'"'
    else : que=que + ' or SIGLA_UNID == "'+unidade+'"'
    i=i+1
print(que)
geolE=geol.query(que)
geolE.to_file('/home/usr/geologiaExt.shp')

```

O mapa resultante é mostrado abaixo, com o mapa geológico completo ao fundo.



7.2.4 Validando o resultado

Quase todo processamento de análise espacial passa pela etapa de validação dos resultados. No nosso caso a quarta etapa consiste em identificar a presença de unidades geológicas novas que não estão ligadas à mineralização ou são mineralizadas de forma secundária. Estas camadas são as mais novas no tempo geológico e podemos nesse caso específico remover as unidades. O script **mapa3mm.py** abaixo efetua a exclusão das cinco unidades mais novas presentes no mapa.

```

#mapa3mm.py
import geopandas as gpd
import pandas as pd
geol=gpd.read_file('/home/usr/geologia.shp',encoding='utf-8')
geol1=gpd.read_file('/home/usr/geologiaF1.shp',encoding='utf-8')

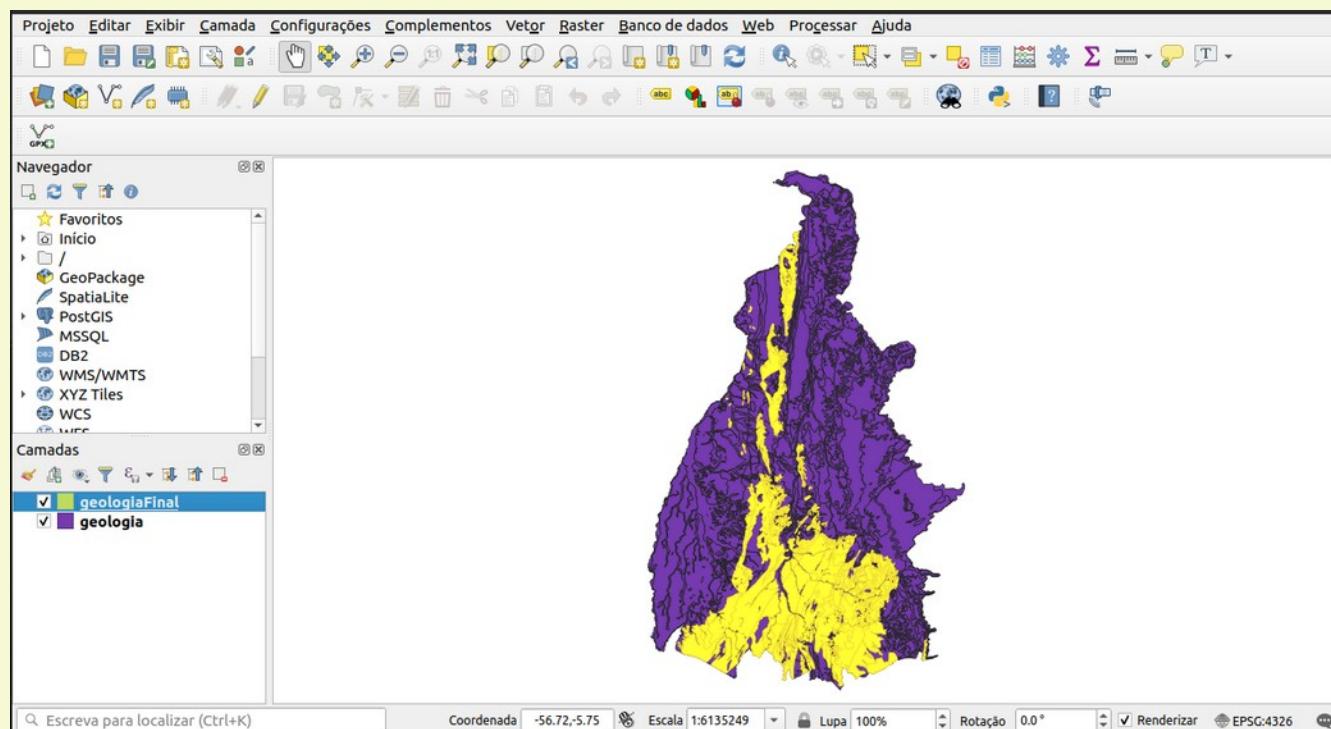
```

```

geol2=gpd.read_file('/home/usr/geologiaF2.shp',encoding='utf-8')
geolF = pd.concat([geol1, geol2], ignore_index=True)
unidades=geolF['SIGLA_UNID'].unique()
i=0
que=''
for unidade in unidades:
    if unidade =='Q2a' or unidade =='N1dl' or unidade =='D23p' or unidade=='P12pf'
or unidade=='P3m':continue
    if i ==0: que='SIGLA_UNID == "'+unidade +"'"
    else : que=que + ' or SIGLA_UNID == "'+unidade+'"'
    i=i+1
print(que)
geolE=geol.query(que)
geolE.to_file('/home/usr/geologiaFinal.shp')

```

O resultado obtido é mostrado abaixo.



7.2.5 Resultado Final

Finalizando a nossa análise espacial, vamos nessa quinta etapa excluir da área potencial as áreas já requeridas por empresas de mineração, as áreas em unidade de conservação e as áreas em terras indígenas, acesse o servidor como fizemos no inicio e grave as camadas ucpi, ucus e índio para efetuarmos a exclusão dessas áreas.

O script final **mapa4mm.py** gerará o nosso produto final como sendo as áreas livres com algum potencial para mineralização primária de metais no estado do Tocantins.

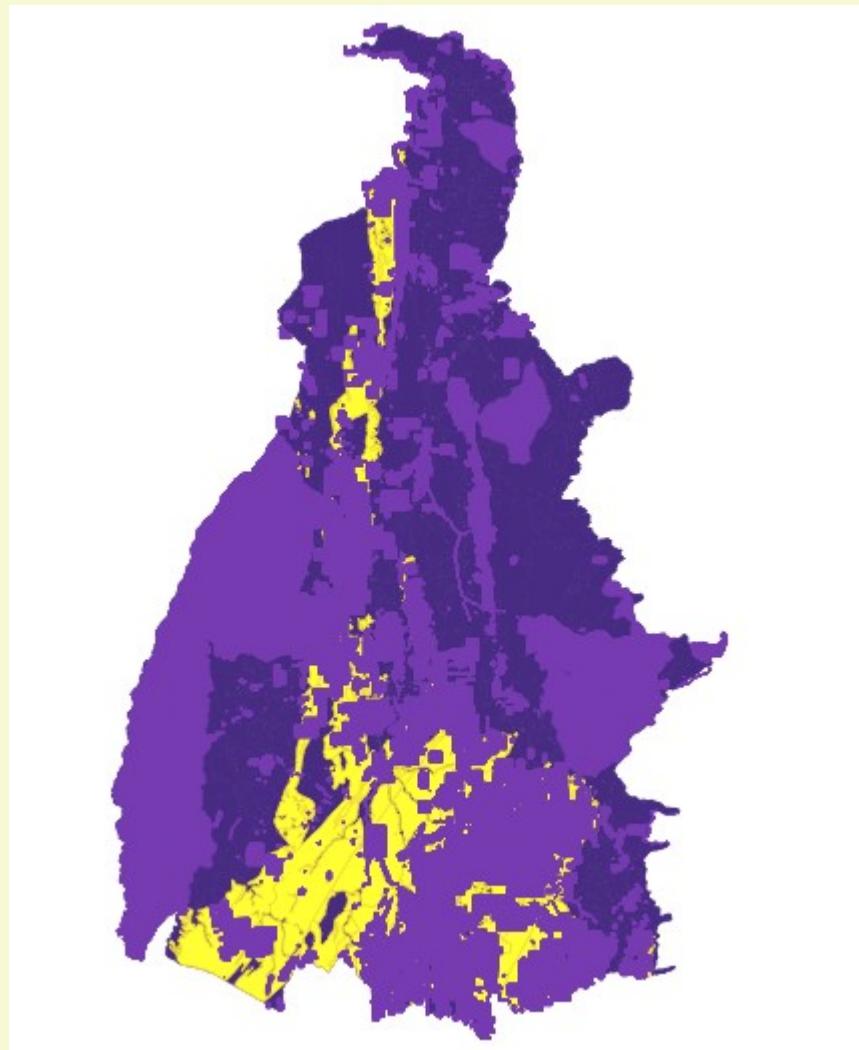
```

#mapa4mm.py
import geopandas as gpd
req=gpd.read_file('/home/usr/requerimentos.shp',encoding='utf-8')
indio=gpd.read_file('/home/usr/indio.shp',encoding='utf-8')
ucus=gpd.read_file('/home/usr/ucus.shp',encoding='utf-8')

```

```
ucpi=gpd.read_file('/home/usr/ucpi.shp',encoding='utf-8')
u1=indio.union(req.unary_union)
u2=u1.union(ucpi)
u3=u2.union(ucus.unary_union)
u3.to_file('/home/usr/mask.shp')
```

Aplicando a máscara das áreas já requeridas e unidades de preservação sobre as áreas potenciais totais temos o resultado das áreas livres com potencial para minerais metálicos no Tocantins é mostrado abaixo em amarelo.



Esta é uma análise espacial superficial e básica com a intenção de mostrar como usar o geopandas. Outros parâmetros deverão ser usados para refinar as áreas potências tais como zonas com estruturas geológicas favoráveis (falhamentos) e filtragem de litologia e grau metamórfico favorável mais refinada.

8. Análise espacial em objeto raster

A utilização e manipulação de imagens raster para efeitos de análise espacial de forma bem básica pode ser feita usando rasterio ou usando as ferramentas do próprio Qgis. Para a manipulação e análise mais robusta e avançada de imagem sugiro o uso de R com a library ‘raster’ em vez de python mas o escopo aqui é python e vamos cobrir a library rasterio e falarmos um pouco sobre GDAL.

8.1 rasterio

Instale rasterio usando pip ou conda.

8.1.1 O básico

Carregando uma imagem raster

Estes são os comandos para se carregar uma imagem raster.

```
>>> import rasterio  
>>> imag = rasterio.open('b2lr.tif')
```

Obtendo informações sobre o objeto raster

Abaixo está uma lista de métodos para se obter informações deste objeto raster.

Obtendo a referência do sistema de coordenadas (CRS) .

```
>>> imag.crs  
CRS.from_epsg(32723)
```

Obtendo os parâmetros de transformação de coordenada da imagem para o sistema de coordenada geográfico.

```
>>> imag.transform  
Affine(20.0, 0.0, 499980.0,  
      0.0, -20.0, 8100040.0)
```

Largura da imagem.

```
>>> imag.width  
5490
```

Altura da imagem.

```
>>> imag.height  
5490
```

Número de bandas da imagem.

```
>>> imag.count  
1
```

Dimensões da imagem em sistema de coordenada geográfica.

```
>>> imag.bounds  
BoundingBox(left=499980.0, bottom=7990240.0, right=609780.0, top=8100040.0)
```

Tipo da imagem (geotiff no caso).

```
>>> imag.driver  
'GTiff'
```

Metadado da imagem.

```
>>> imag.meta
{'driver': 'GTiff', 'dtype': 'uint16', 'nodata': None, 'width': 5490, 'height': 5490, 'count': 1, 'crs': CRS.from_epsg(32723), 'transform': Affine(20.0, 0.0, 499980.0, 0.0, -20.0, 8100040.0)}
```

Tipo do objeto.

```
>>> type(imag)
<class 'rasterio.io.DatasetReader'>
```

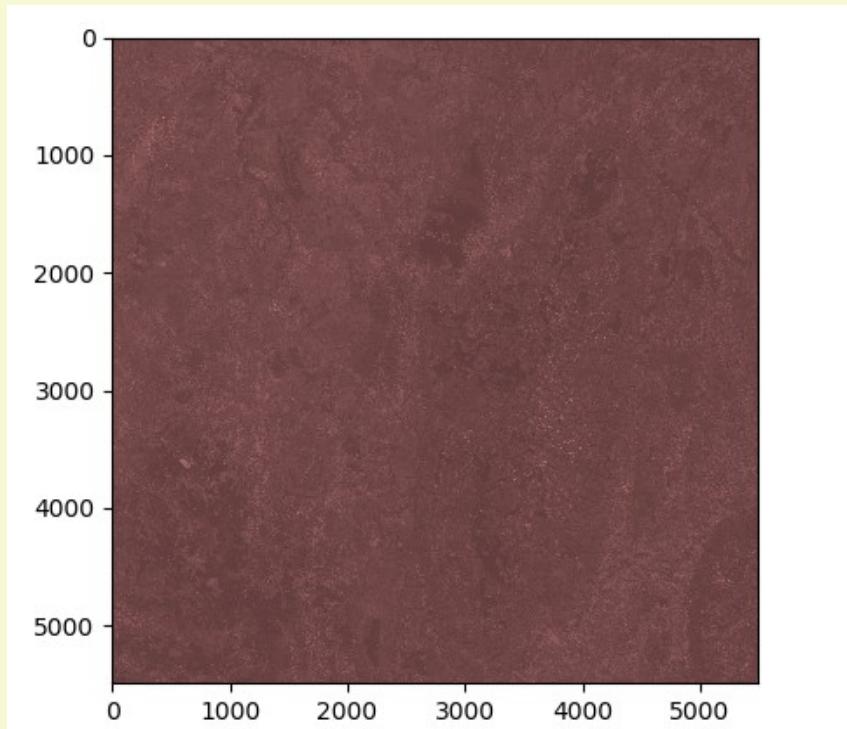
8.1.2 Carregando e plotando uma banda da imagem

Cada banda da imagem é um objeto ndarray e acessamos esta banda usando o método `read(número da banda)`.

```
>>> banda2=imag.read(1)
>>> type(banda2)
<class 'numpy.ndarray'>
```

Podemos visualizar esta banda usando pyplot.

```
>>> from matplotlib import pyplot
>>> pyplot.imshow(banda2, cmap='pink')
>>> pyplot.show()
```



Alguns valores estatísticos da banda podem ser vistos usando:

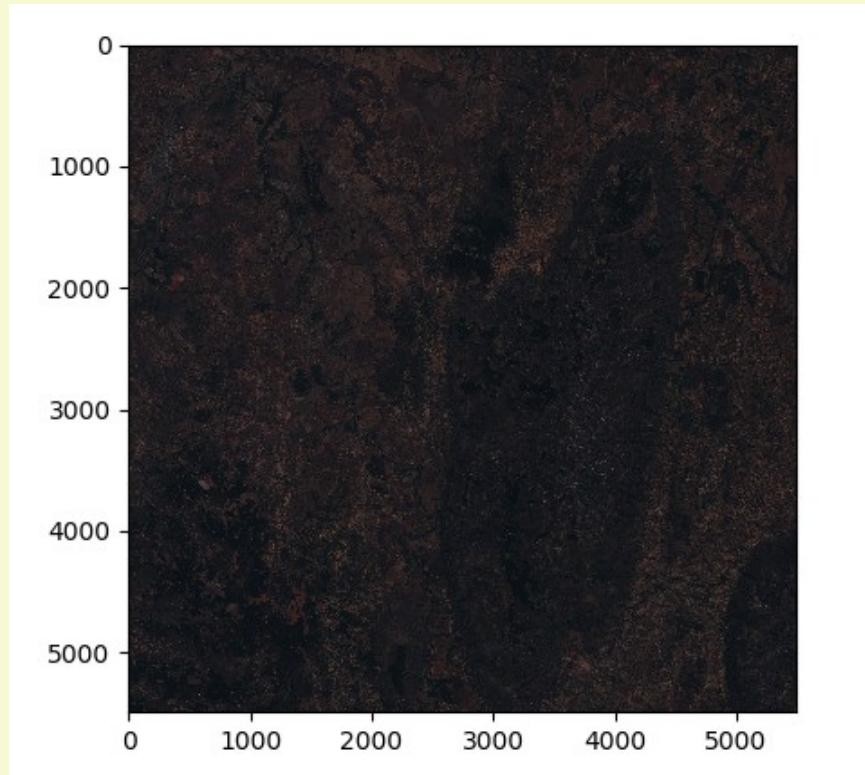
```
>>> banda2.min()
0
>>> banda2.max()
7938
>>> banda2.mean()
972.3386621145916
```

8.1.3 Composição colorida RGB

Para gerar uma imagem RGB com rasterio podemos usar o seguinte (bem lento)

```
>>> import rasterio  
>>> import numpy as np  
>>> import matplotlib.pyplot as plt  
>>> r=rasterio.open('b4lr.tif').read(1)  
>>> g=rasterio.open('b3lr.tif').read(1)  
>>> b=rasterio.open('b2lr.tif').read(1)  
>>> def normaliza(array):  
...     array_min, array_max = array.min(), array.max()  
...     return (array - array_min) / (array_max - array_min)  
...  
>>> nr=normaliza(r)  
>>> nb=normaliza(b)  
>>> ng=normaliza(g)  
>>> rgb = np.dstack((nr, ng, nb))  
>>> plt.imshow(rgb)  
>>> plt.show()
```

Que gerará a seguinte imagem RGB:



8.1.4 Operações entre bandas

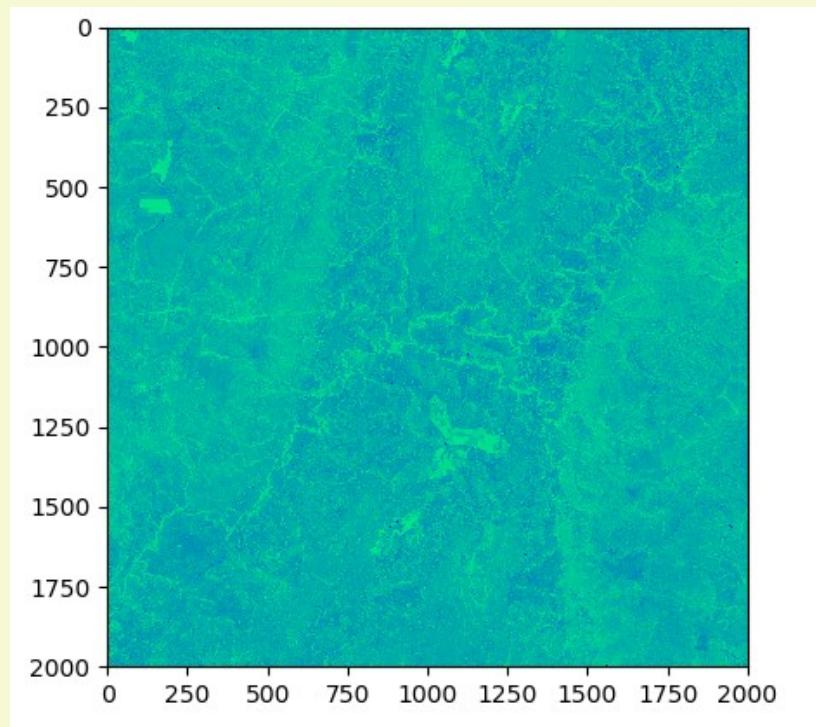
A álgebra de bandas com rasterio pode ser efetuada com relativa facilidade, vamos mostrar abaixo como efetuar uma razão entre bandas para gerar uma imagem NDVI.

```

>>> import rasterio
>>> from rasterio.plot import show
>>> import numpy as np
>>> r= rasterio.open('b4.tif').read(1)
>>> nir= rasterio.open('b8.tif').read(1)
>>> r=r.astype(float)
>>> nir=nir.astype(float)
>>> np.seterr(divide='ignore', invalid='ignore')
>>> ndvi = np.empty(r.shape, dtype=rasterio.float32)
>>> checa = np.logical_or ( r > 0, nir > 0 )
>>> ndvi = np.where ( checa, (nir - r) / ( nir + r ), -999 )
>>> show(ndvi, cmap='winter')

```

O resultado é mostrado abaixo:



8.2 GDAL

O GDAL é uma ferramenta de software livre que manipula e modifica formatos geoespaciais do tipo raster e vetorial. Como uma library ela apresenta modelo simples de abstração de dados raster ou vetorial para a aplicação que o chama em todos os formatos suportados. O GDAL também vem com uma grande variedade de comandos (programas) úteis para modificar e processar os dados.

Os programas de linha de comandos voltados para raster mais usados são:

gdalinfo: Lista a informação sobre o dado raster.

gdal_translate: Converte dados raster em diferentes formatos.

gdaladdo: Cria ou recria uma visualização de imagem.

gdalwarp: Ferramenta de reprojeção ou ajuste de imagem.

gdaltindex: Cria um shapefile como índice de fragmentos (tile) de raster.

gdalbuildvrt: Cria um VRT (raster virtual em xml) a partir de uma lista de dados.

gdal_contour: Cria linhas de contorno vetoriais a partir de um modelo digital de elevação DEM.

gdaldem: Ferramenta para analisar e visualizar DEM.

rgb2pct: Converte imagem RGB de 24bits para imagem RGB de 8bits paletada.

pct2rgb: Converte o oposto do citado acima.

gdal_merge: Cria mosaico de um conjunto de imagens.

gdal_rasterize: Introduz geometrias vetoriais em um raster.

gdaltransform: Transforma o sistema de coordenadas.

nearblack: Converte bordas quase pretas/brancas para preto.

gdal_grid: Cria um gride regular a partir de dados pontuais espalhados.

gdal_polygonize: Produz uma camada com um polígono representando um raster.

gdal_sieve: Remove polígonos rasters menores.

gdal_fillnodata: Preenche as áreas sem dados de um raster por interpolação de seus limites.

gdal_edit: Edita várias informações de um dado GDAL existente.

gdal_calc.py: Calcula operações com imagens raster com sintaxe do numpy.

gdal_pansharpen.py: Executa uma operação de delineamento na imagem raster.

gdal-config: Determina várias informações sobre a instalação do GDAL.

8.2.1 Conceitos básicos do GDAL

A informação abaixo foi extraída e traduzida do site do Derek Watkins (<https://github.com/dwtkns/gdal-cheat-sheet>) e introduz alguns exemplos de como usar os comandos do GDAL.

Obtendo informações sobre uma imagem raster.

```
gdalinfo arquivo.tif
```

Lista os drivers de formatos raster suportados.

```
gdal_translate --formats
```

Convertendo formatos raster

```
gdal_translate -of "GTiff" arq_entrada.grd arq_saida.tif
```

Convertendo bandas 16-bit (Int16 ou UInt16) no tipo Byte

```
gdal_translate -of "GTiff" -co "COMPRESS=LZW" -scale 0 65535 0 255 -ot Byte
entrada_uint16.tif saida_byte.tif
```

Reprojetando um raster:

```
gdalwarp -t_srs "EPSG:102003" arq_entrada.tif arq_saida.tif
```

Tenha certeza de usar a opção -r bilinear se estiver reprojetando DEM para prevenir bandamentos estranhos.

Georeferenciamos uma imagem não projetada com coordenadas delimitantes conhecidas:

```
gdal_translate -of GTiff -a_ullr <lon_sup_esq> <lat_sup_esq> <lon_inf_dir>
<lat_inf_dir> -a_srs EPSG:4269 arq_entrada.png arq_saida.tif
```

Recortando raster com uma caixa de extensão

```
gdalwarp -te <x_min> <y_min> <x_max> <y_max> arq_entrada.tif saida_recortada.tif
```

Recortando raster assinalando SHP/NoData para os pixels além da extensão do polígono.

```
gdalwarp -dstnodata <valor_nodata> -cutline input_polygon.shp entrada.tif  
saída_recortada.tif
```

Recortando dimensões de um raster para a caixa de extensão de um vetor.

```
gdalwarp -cutline recortador.shp -crop_to_cutline entrada.tif saída_recortada.tif
```

Unindo rasters.

```
gdal_merge.py -o unido.tif entrada1.tif entrada2.tif
```

Alternativamente,

```
gdalwarp entrada1.tif entrada2.tif unido.tif
```

Ou, para preservar os valores nodata :

```
gdalwarp input1.tif input2.tif merged.tif -srcnodata <nodata_value> -dstnodata  
<merged_nodata_value>
```

Empilhando bandas simples cinza em um arquivo de bandas RGB onde LC81690372014137LGN00 é um arquivo Landsat 8 ID e B4, B3 e B2 correspondem às bandas R,G,B respectivamente:

```
gdal_merge.py -co "PHOTOMETRIC=RGB" -separate LC81690372014137LGN00_B{4,3,2}.tif -o  
LC81690372014137LGN00_rgb.tif
```

Arrumando um arquivo RGB TIF que desconhece que é RGB.

```
gdal_merge.py -co "PHOTOMETRIC=RGB" entrada.tif -o saída_rgb.tif
```

Exportando um raster para o Google Earth (arquivo kmz).

```
gdal_translate -of KMLSUPEROVERLAY entrada.tif saída.kmz -co FORMAT=JPEG
```

Operações algébricas com Raster.

Média de dois rasters:

```
gdal_calc.py -A entrada1.tif -B entrada2.tif --outfile=resultado_media.tif --  
calc="(A+B)/2"
```

Adicionando dois rasters:

```
gdal_calc.py -A entrada1.tif -B entrada2.tif --outfile=resultado_soma.tif --  
calc="A+B"
```

Criando sombreamento de relevo (hillshade) a partir de um DEM.

```
gdaldem hillshade -of PNG dem.tif hillshade.png
```

Mudando a direção da iluminação no hillshade.

```
gdaldem hillshade -of PNG -az 135 entrada.tif hillshade_az135.png
```

Usando uma correção de escala vertical para metros se o raster de entrada for em graus.

```
gdaldem hillshade -s 111120 -of PNG entrada_WGS1984.tif hillshade.png
```

Aplicando uma rampa de cor ao DEM.

Primeiro crie o arquivo rampa_cor.txt com (Elevação, R, G, B)

```
0 110 220 110  
900 240 250 160  
1300 230 220 170  
1900 220 220 220  
2500 250 250 250
```

Então aplique essas cores ao DEM usando.

```
gdaldem color-relief DEM.tif rampa_cor.txt DEM_colorido.tif
```

Criando um sombreamento de plano inclinado a partir de um DEM

Primeiro crie um raster de plano inclinado a paratir do DEM usando:

```
gdaldem slope dem.tif slope.tif
```

Depois crie um arquivo cor_slope.txt onde (ângulo, R, G, B)

```
0 255 255 255  
90 0 0 0
```

Finalmente, colorir o raster de plano inclinado baseado nos ângulos do arquivo cor_slope.txt usando:

```
gdaldem color-relief slope.tif cor_slope.txt saida_slopeshade.tif
```

Reamostrando (mudando a dimensão) do raster

```
gdalwarp -ts <largura> <altura> -r cubic dem.tif reamostrado_dem.tif
```

Entrando 0 para a largura e altura irá estimará baseado nas dimensões correntes.

Alternativamente podemos usar porcentagens.

```
gdal_translate -outsize 10% 10% -r cubic dem.tif reamostrado_dem.tif
```

Criando um raster a partir de um dado vetorial (rasterizar)

```
gdal_rasterize -b 1 -i -burn -32678 -l nomedacamada entrada.shp saida.tif
```

Extraindo raster como um polígono

```
gdal_polygonize.py entrada.tif -f "GeoJSON" saida.json
```

Criando curvas de contorno a partir de um DEM

```
gdal_contour -a elev -i 50 entrada_dem.tif saida_contorno.shp
```

8.2.2 Usando GDAL via chamada de sistema do python

A maneira mais eficiente até o momento de lidar com manipulação e análise de raster no Qgis é através da execução dos programas GDAL diretamente na linha de comando ou usando chamada de sistema via script python. O Qgis faz isso constantemente quando usamos o menu **Raster -> Análise**. Todos os processamentos são efetuados via plugin que faz uma chamada ao programa GDAL.

Vamos fazer a diferença normalizada do índice de vegetação (NDVI) que mostra a saúde da vegetação com base nas respostas espectrais entre as bandas do vermelho (Red) e infravermelho próximo (NIR) que no Sentinel2 correspondem às bandas 4 e 8 respectivamente.

O script **executaGdal.py** abaixo faz uma chamada ao GDAL e cria o arquivo dvi.tif usando a fórmula definida por calc.

```
# executarGdal.py  
import os  
cmd='gdal_calc.py -outfile dvi.tif -A b8.tif -B b4.tif --calc="(A-B) / (A+B)" -overwrite'  
os.system(cmd)
```

Você poderia também executar o programa **gdal_calc.py** diretamente na linha de comando com:

```
gdal_calc.py -outfile dvi.tif -A b8.tif -B b4.tif --calc="(A-B) / (A+B)" -overwrite
```

Pratique com os outros programas GDAL, mais detalhes sobre GDAL podem ser encontrados em <https://gdal.org/programs/index.html>