

# Uma rápida Introdução ao QGIS e PyQGIS



André Luiz Lima Costa

Licença:

[https://creativecommons.org/licenses/by/4.0/deed.pt\\_BR](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:

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

VISITE:

[amazeone.com.br](https://amazeone.com.br)

## Sumário

<b>1. QGIS .....</b>	4
1.0 Introdução .....	4
1.1 Instalando e Iniciando o QGIS .....	4
1.2 Carregando dados Vetoriais .....	5
1.3 Carregando dados Raster .....	15
1.4 Criando Dados Vetoriais.....	20
1.5 Criando Raster a partir de Pontos Vetoriais.....	24
<b>2. Fundamentos de Python .....</b>	28
2.0 A linguagem Python – Introdução.....	28
2.1 Fundamentos da linguagem Python .....	29
2.2 Controles de fluxo .....	35
2.3 Funções .....	37
2.4 Módulos .....	39
2.5 Pandas .....	39
2.6 Gráficos .....	42
<b>3. Usando Python no QGIS .....</b>	44
3.1 Primeiros passos, noções de Classes.....	44
3.2 Interagindo com informações de objetos da classe Vector .....	48
3.3 Interagindo com informações de objetos da classe Raster .....	56
3.4 Criando objeto vetorial .....	62
3.5 Criando objeto raster .....	68
<b>4. Executando python scripts fora do Qgis .....</b>	71

## 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 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 3.16 LTR (Hannover) que é versão estável mais recente. Versões mais recentes de desenvolvimento já lançadas são a 3.22 (Białowieża) . 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*.

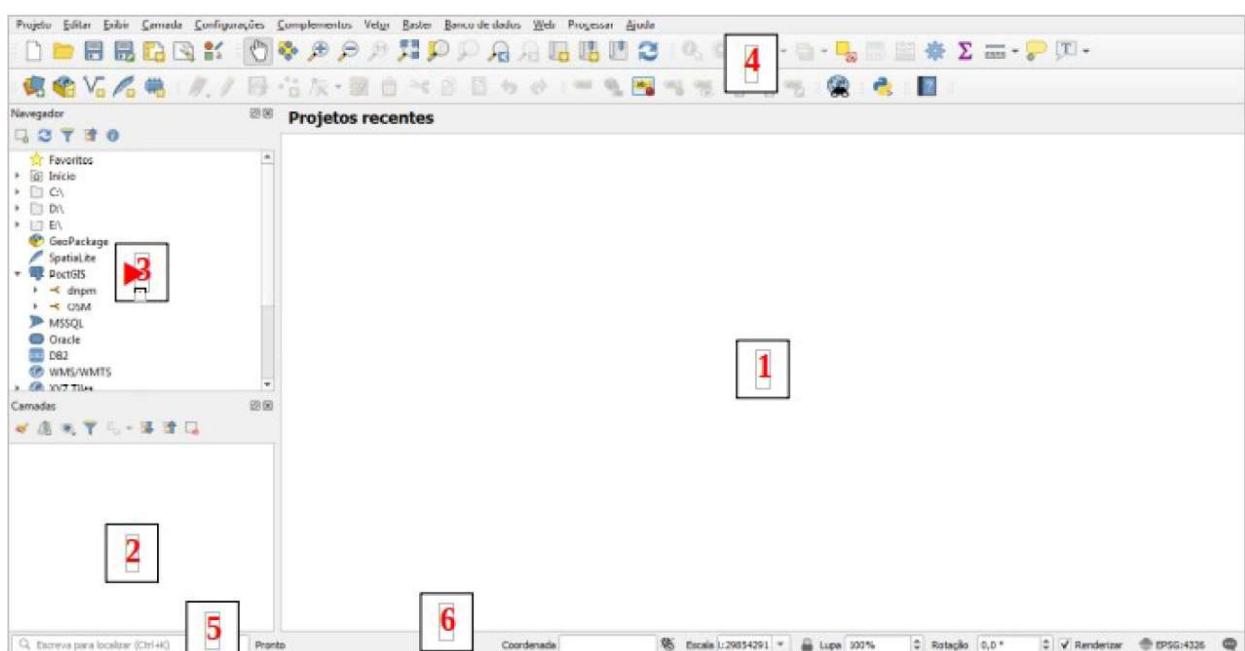
Veremos nessa apostila a integração da linguagem Python com o QGIS com o objetivo de automatização de processos e análises espaciais de dados bem como a integração com banco de dados geoespaciais.

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](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 mostrado na 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 – À 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 os 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 etc. podem ser vistos 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.

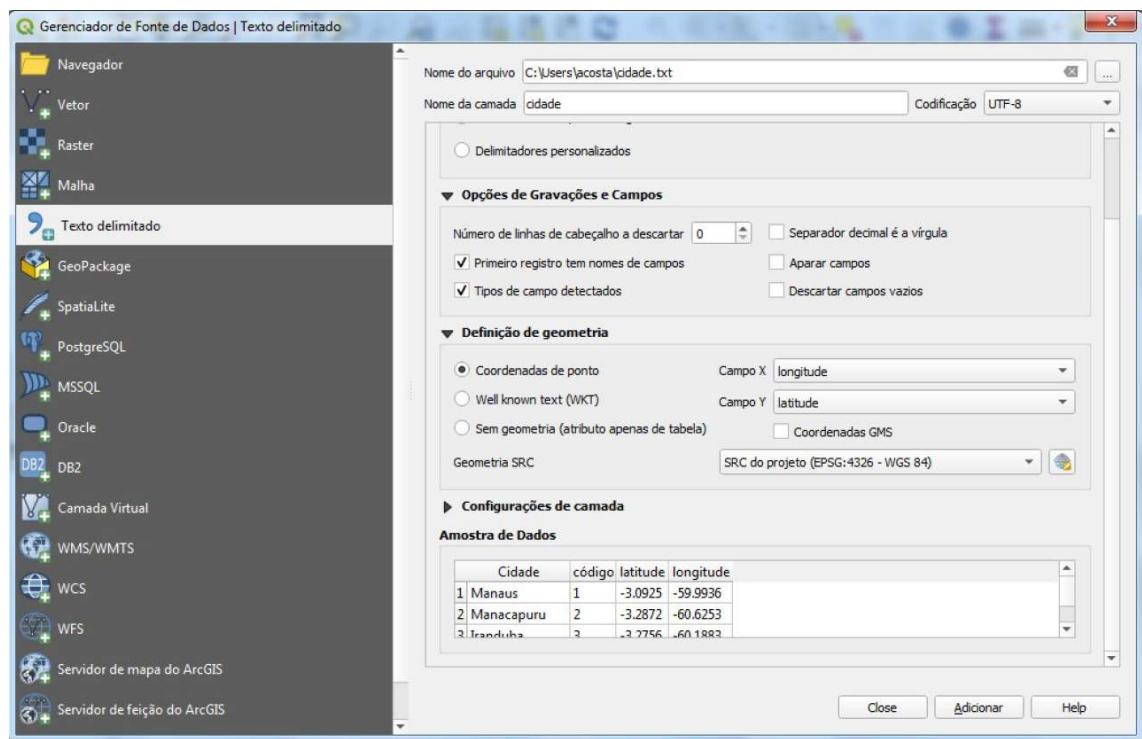
### Dados no formato texto

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

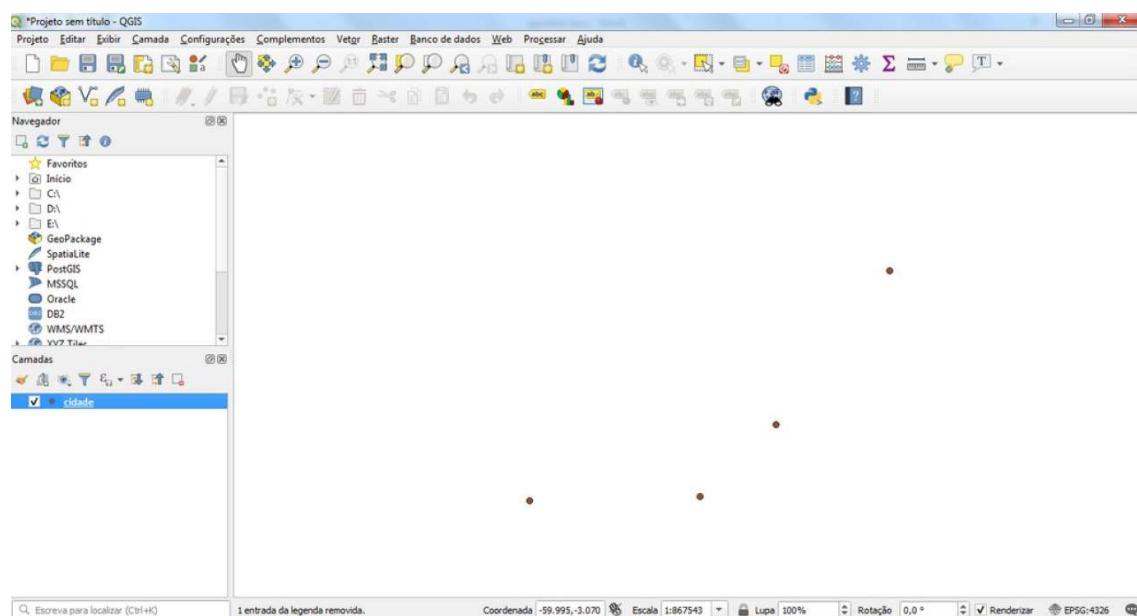
```
Cidade,código,latitude,longitude
Manaus,1,-3.0925,-59.9936
Manacapuru,2,-3.2872,-60.6253
Iranduba,3,-3.2756,-60.1883
Rio Preto de Eva,4,-2.6968,-59.7014
```

- 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 usando o botão ... 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**.

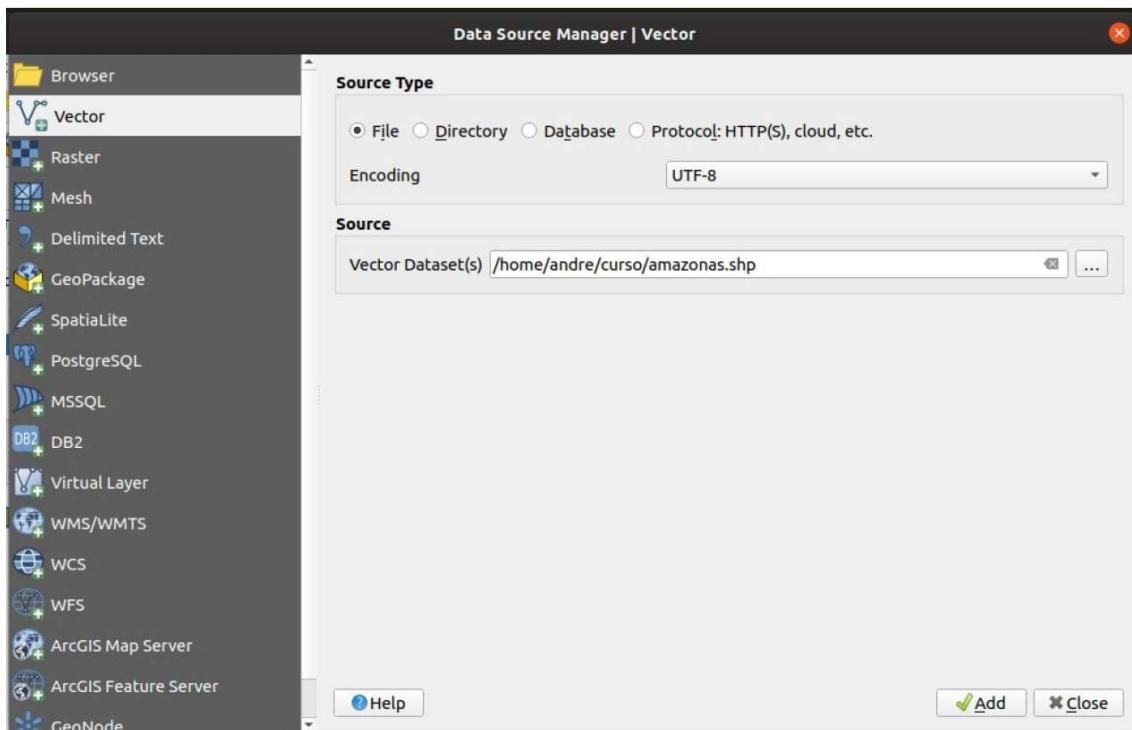


A camada cidade será carregada.

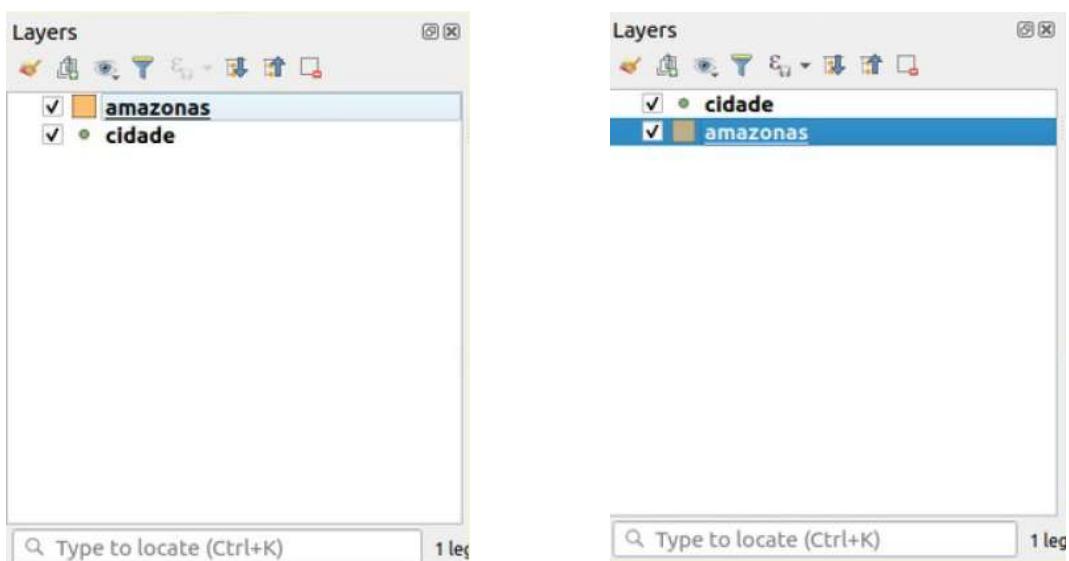


## Dados no Formato GIS (ESRI Shapefile, GMT, MapInfo, etc)

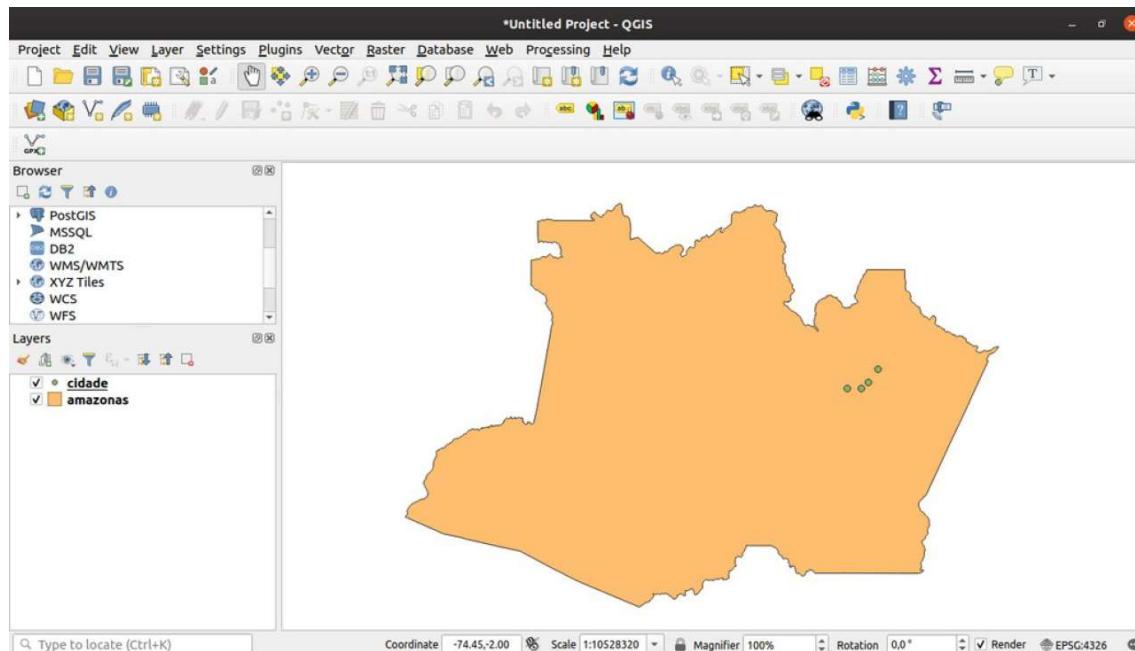
- Vá no menu **Camada > Adicionar Camada > Vetorial**.
  - Selecione no lado direito a opção Vector.
  - Navegue até o local do arquivo criado acima no botão ... e selecione o arquivo **amazonas.shp**.
- Todos os campos devem ficar conforme a imagem abaixo e em seguida clique em **Adicionar** e depois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



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



Abaixo vemos o resultado.

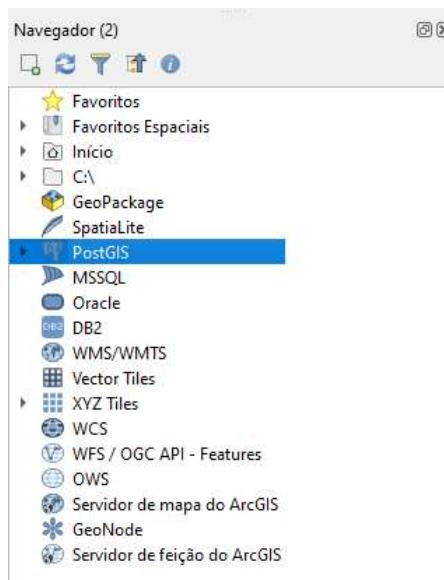


Podemos gravar o nosso projeto com o nome de primeiro. Vá até o menu **Projeto > Salvar**. Na janela que aparece escreva o nome '**primeiro**'. Pronto, o projeto está salvo.

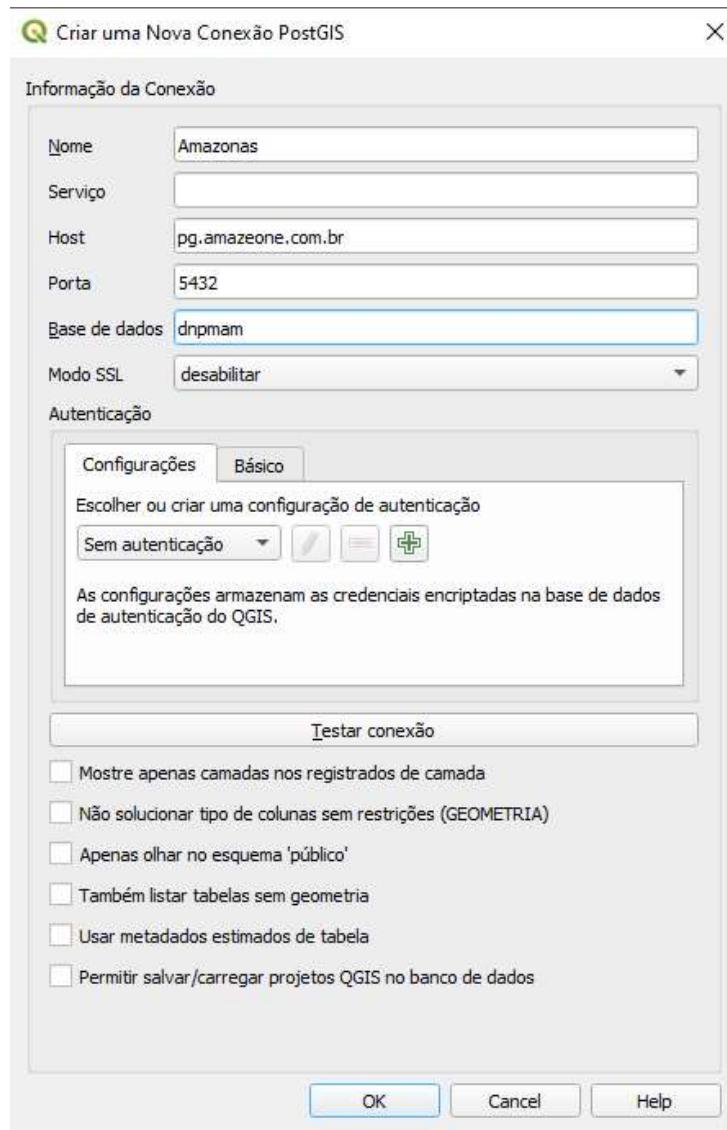
#### [Abrindo Dados de Fontes Remotas \(Banco de dados em servidores remotos\)](#)

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. Também dados de ArcGISMapServer, WFS, XYZ Tiles, etc.

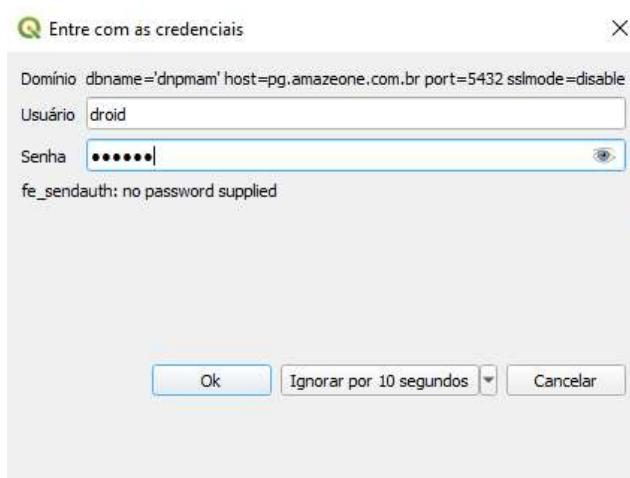
Vamos aqui mostrar como acessar um banco de dados Postgis-PostgreSQL e carregar um objeto espacial. Do lado esquerdo, no painel 'Navegador'. Clique com o botão direito do mouse e selecione **Nova Conexão...**



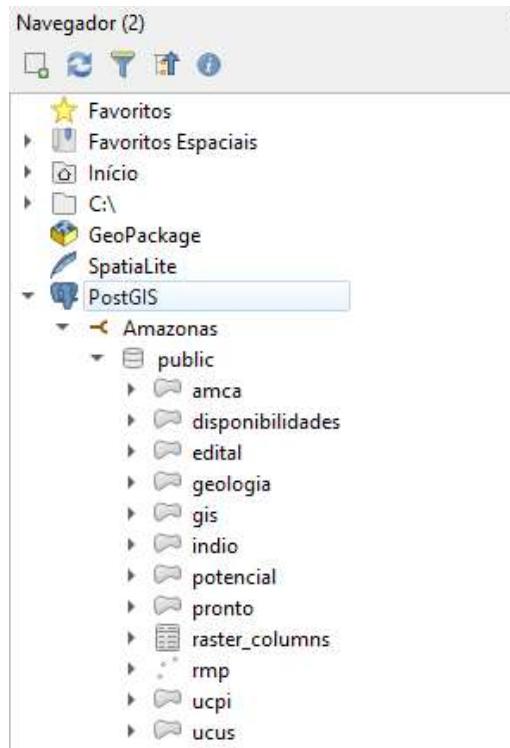
Preencha os campos conforme mostrados abaixo:



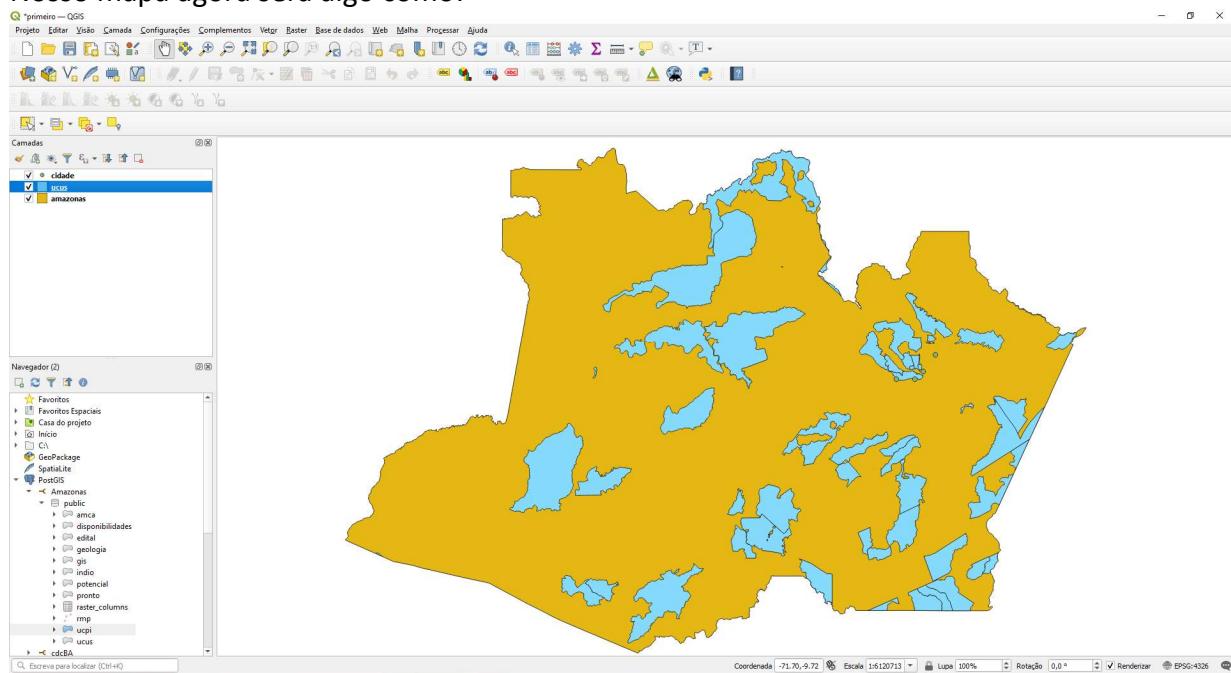
A janela abaixo vai aparecer. Entre com **droid** como usuário e **devcor** como senha, Clique **OK**.



Clique na seta para baixo em **public** e selecione, clicando duas vezes no objeto **ucus** para carregar ele:



Nosso mapa agora será algo como:



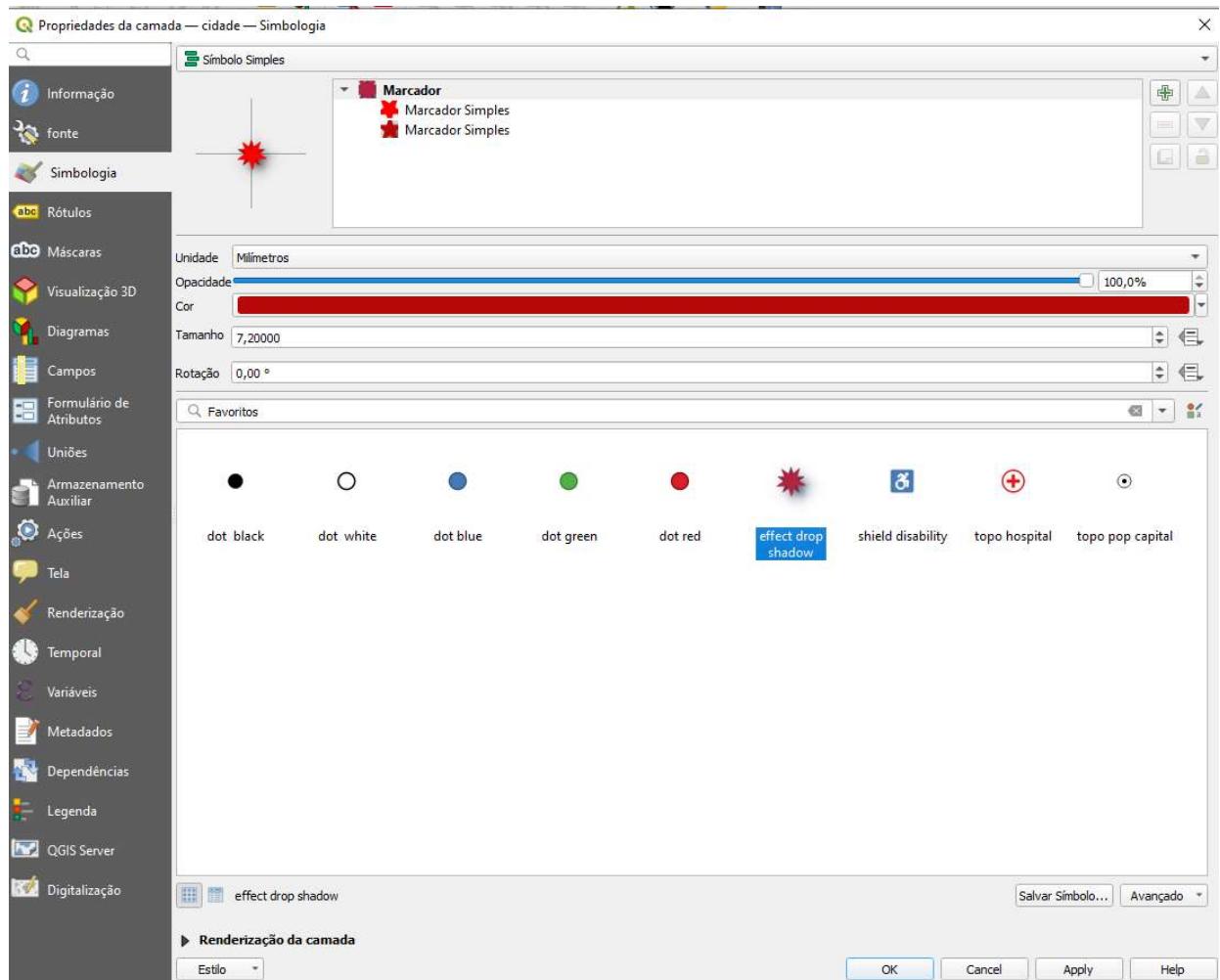
Vimos aqui como carregar objetos de dados espaciais de diversos formatos de maneira bem simples para dentro do QGIS. Salve novamente o projeto. Ao abrir novamente o projeto as credenciais do banco de dados devem ser inseridas novamente (usuário **droid** e senha **devcor**) para carregar o dado remoto.

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.

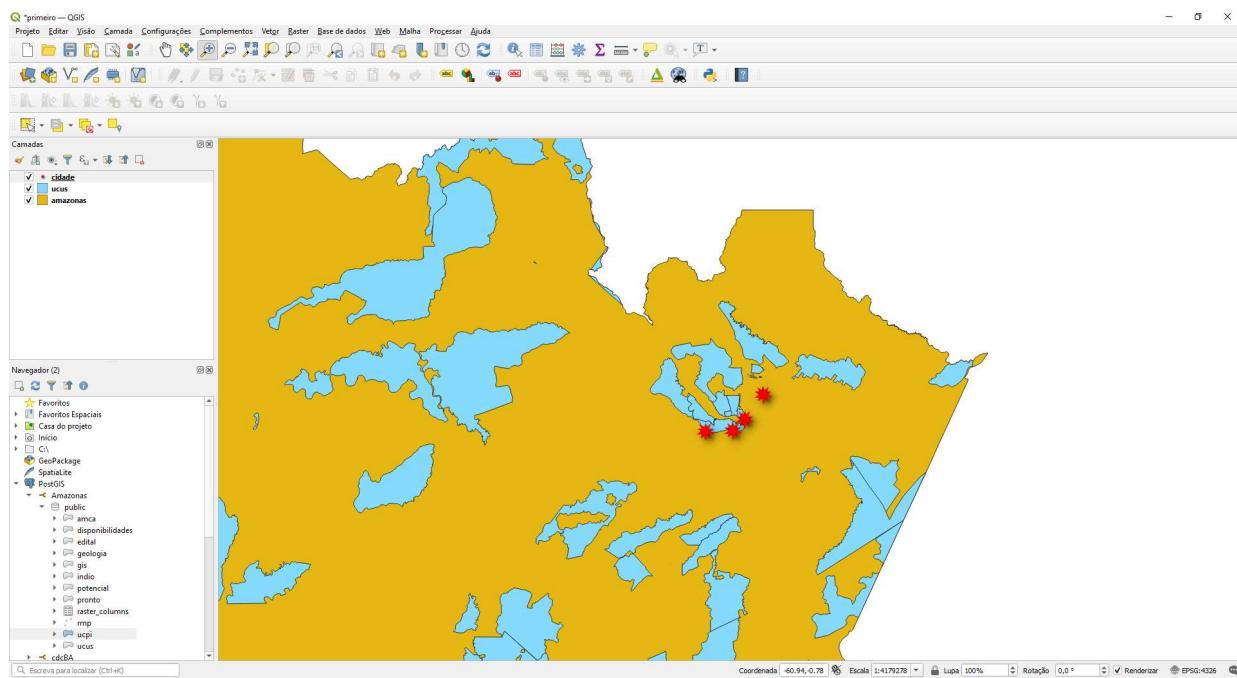
### Propriedades e atributos dos objetos espaciais

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

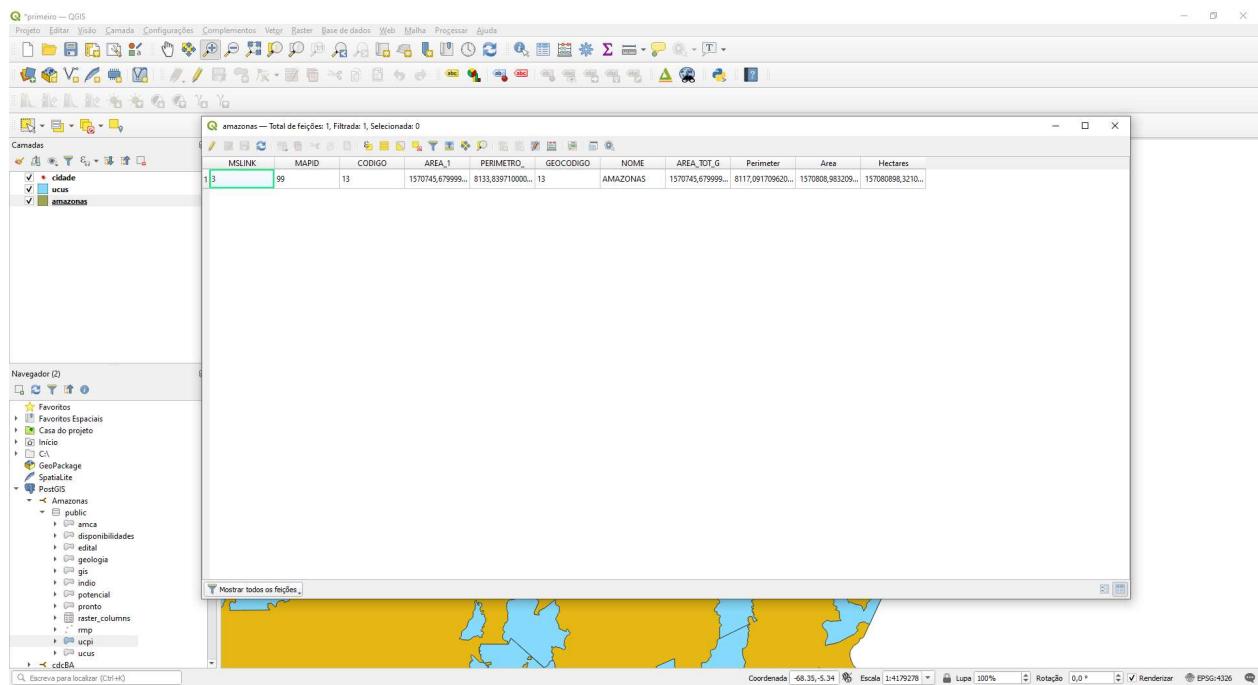


Modificamos os parâmetros de cada objeto selecionando diferentes símbolos, cores, espessuras e padrões, podemos também personalizar símbolos usando o botão **Salvar Símbolo**.

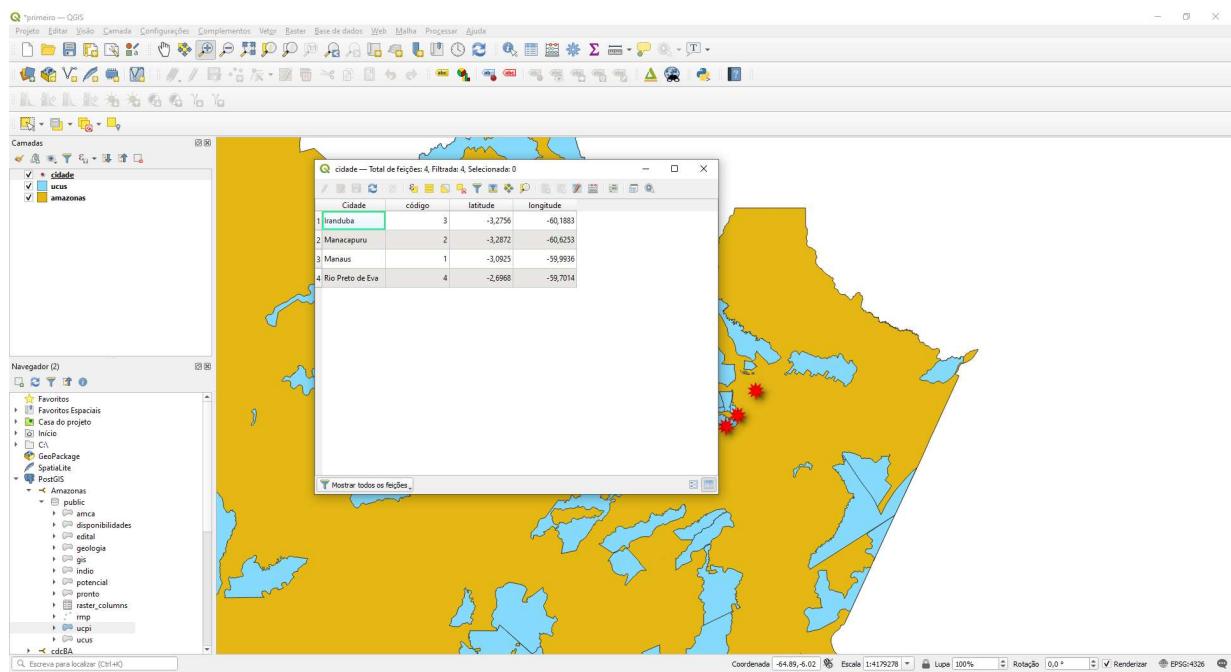
Vamos efetuar um exercício e transformar a aparência de cada objeto até obtermos um resultado semelhante ao da imagem abaixo:



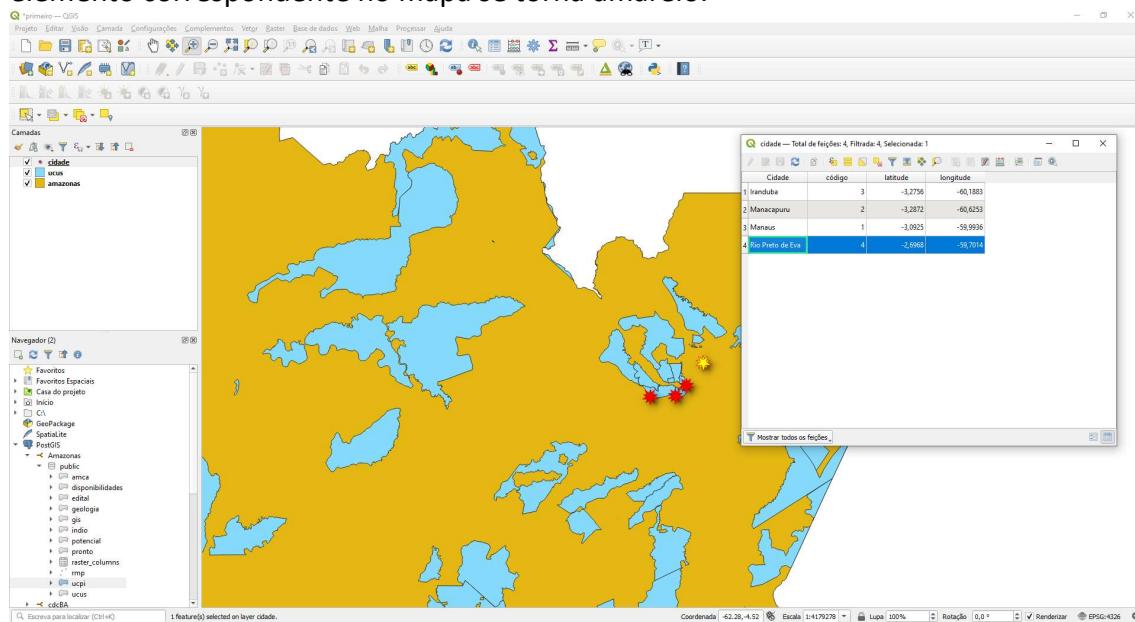
Abaixo mostramos como ver a tabela de atributos das camadas abertas. Obra o projeto **primeiro.qgz** caso não esteja aberto e no painel de camadas selecione a camada **amazonas**, vá no menu **Camada > Abrir tabela de Atributos** ou pressione F6. A seguinte janela aparecerá.



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, ajuste as janelas conforme abaixo para mostrar como funciona o processo de selecionar elementos:



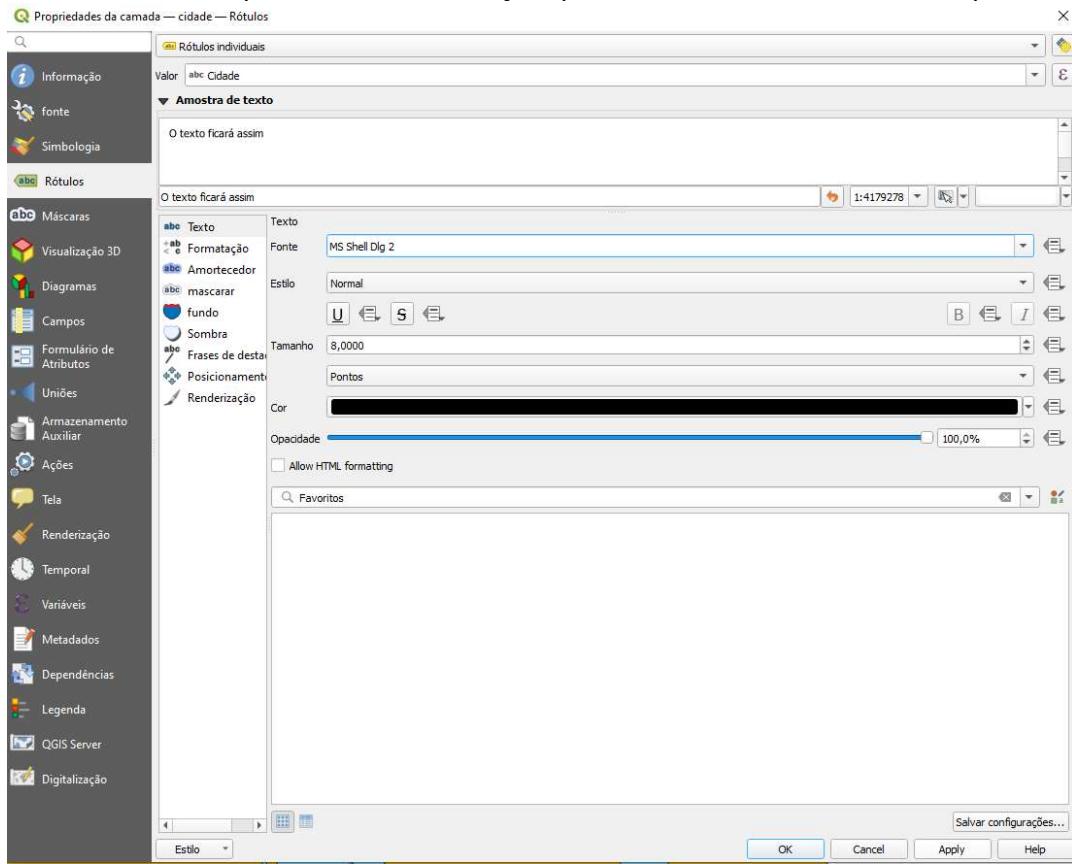
Ao clicarmos na numeração no canto esquerdo da tabela a linha é selecionada e o elemento correspondente no mapa se torna amarelo.



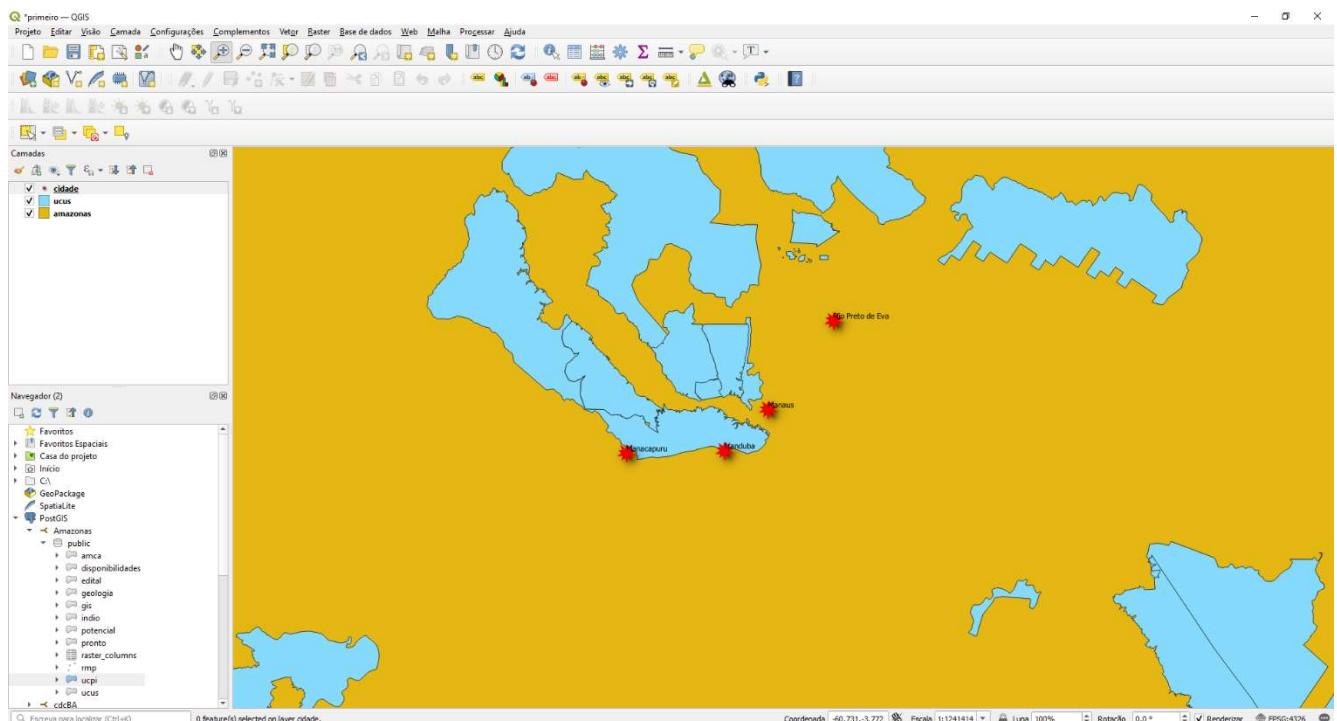
Abaixo vemos as opções de seleção da barra de ferramentas da janela da tabela de dados.

-  **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

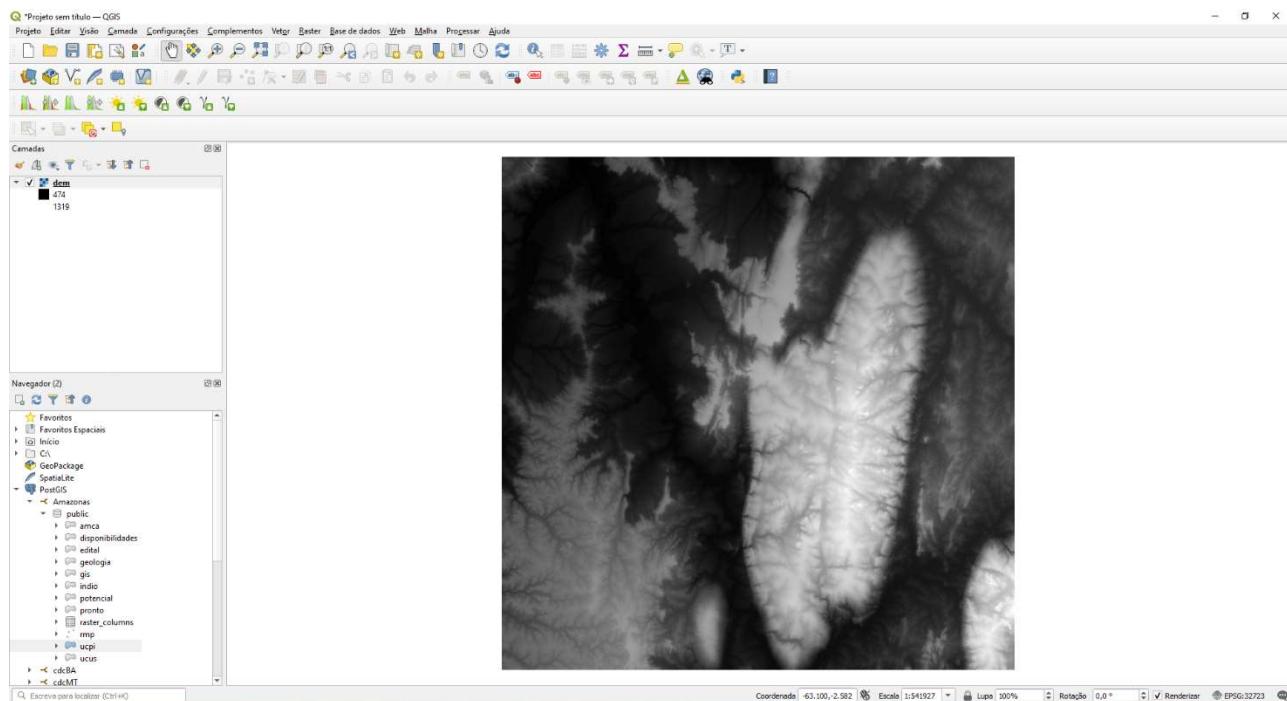
QGIS trabalha com dados no formato raster (imagem) também. Dados do tipo raster são uma representação de quantidades ou grandezas que variam ou não em intervalos regulares (grid), em vez de uma grandeza única pontual ou ao longo de uma linha ou numa área específica como é um dado vetorial.

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.

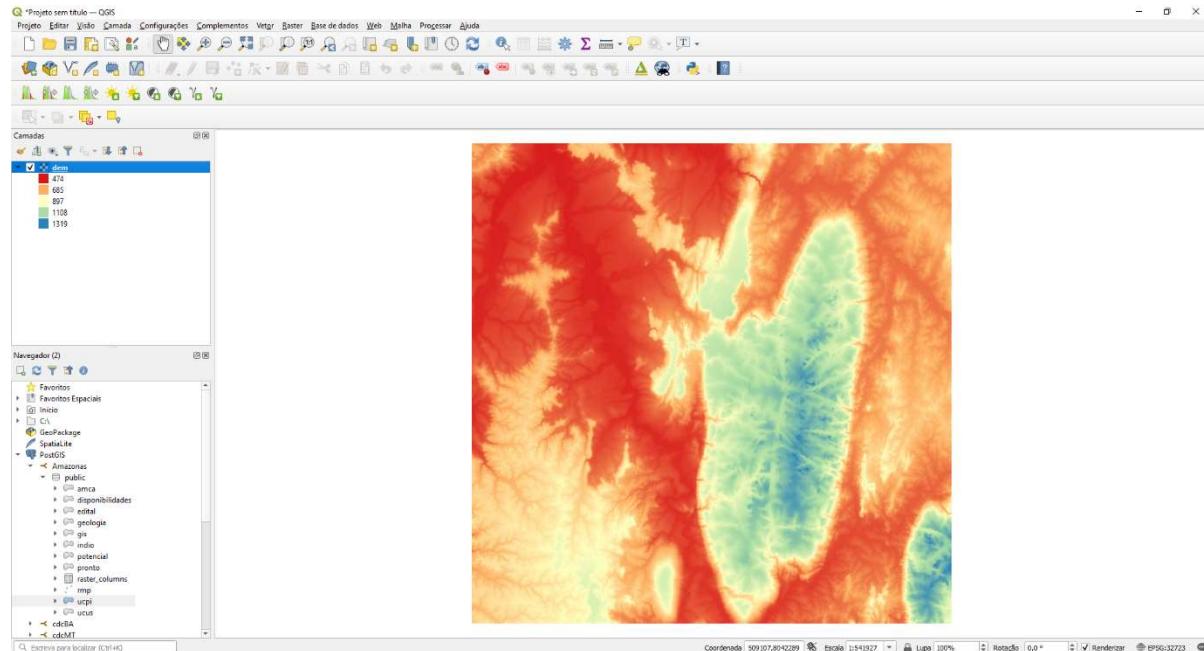
Dados pontuais do tipo vetorial podem ser convertidos em dados do tipo raster usando interpolações matemáticas ou estatísticas que calcula ou estima a distribuição dos valores em intervalos regulares de acordo com o grid a ser criado.

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.

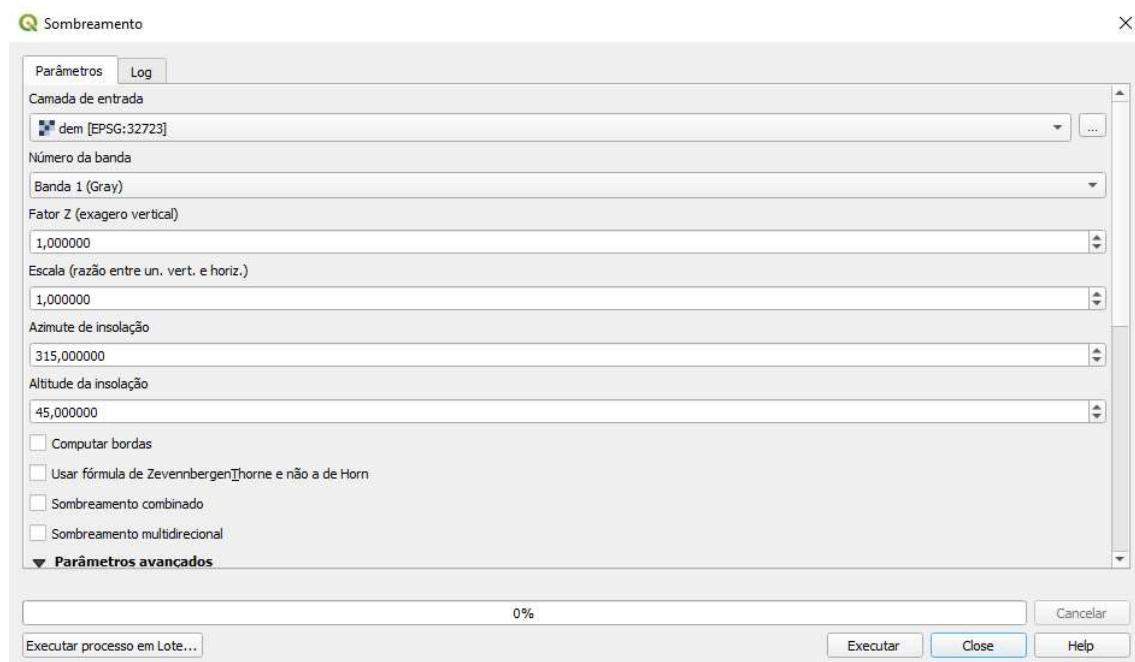
Abra o QGIS e carregue a imagem raster dem.tif **Camada > Adicionar Camada > Raster** ou **Ctrl+Shift +r** e proceda acionando ... para navegar ao local do arquivo **dem.tif**. Em seguida clique em **Adicionar** e **Close**. A objeto raster deverá ser carregado conforme mostrado abaixo.

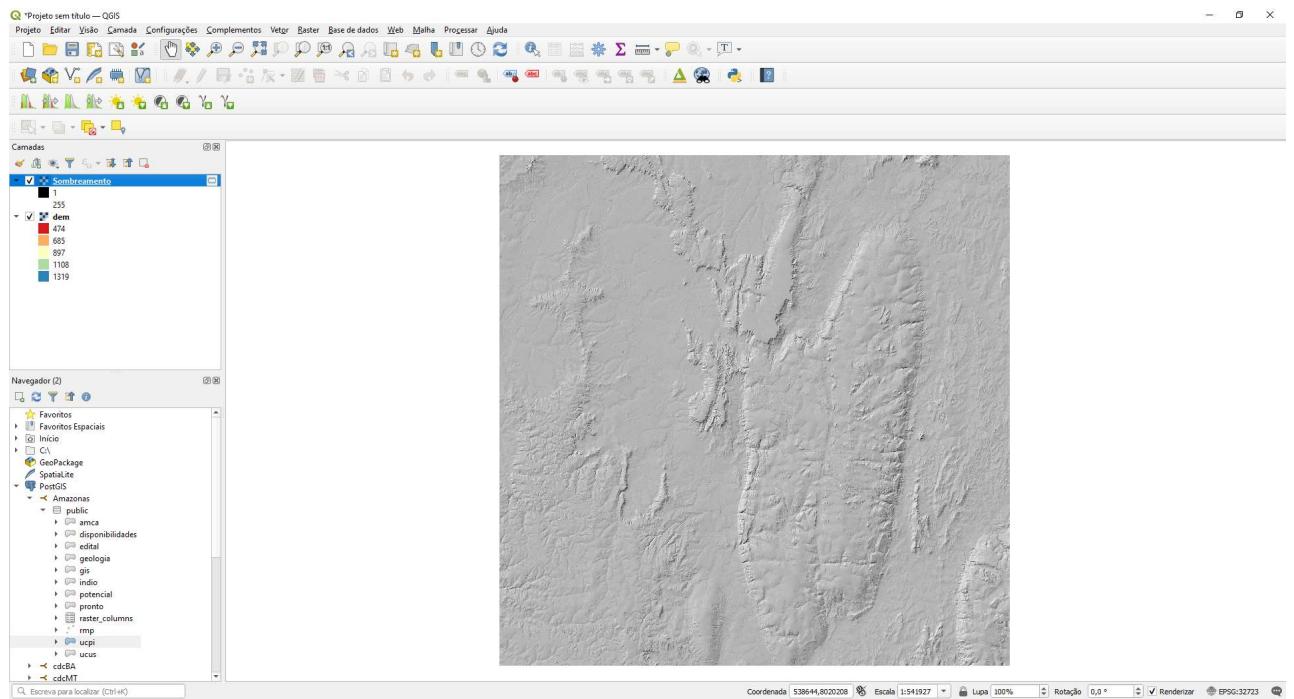


No painel Camadas clique duas vezes em dem e vamos modificar as cores de cinza para uma rampa de cor já predefinida. Na janela que se abriu selecione **Simbologia** no painel da esquerda e selecione **Paletizado/Valores Únicos** no campo **Tipo de renderização**. Selecione **Spectral** no campo Gradiente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



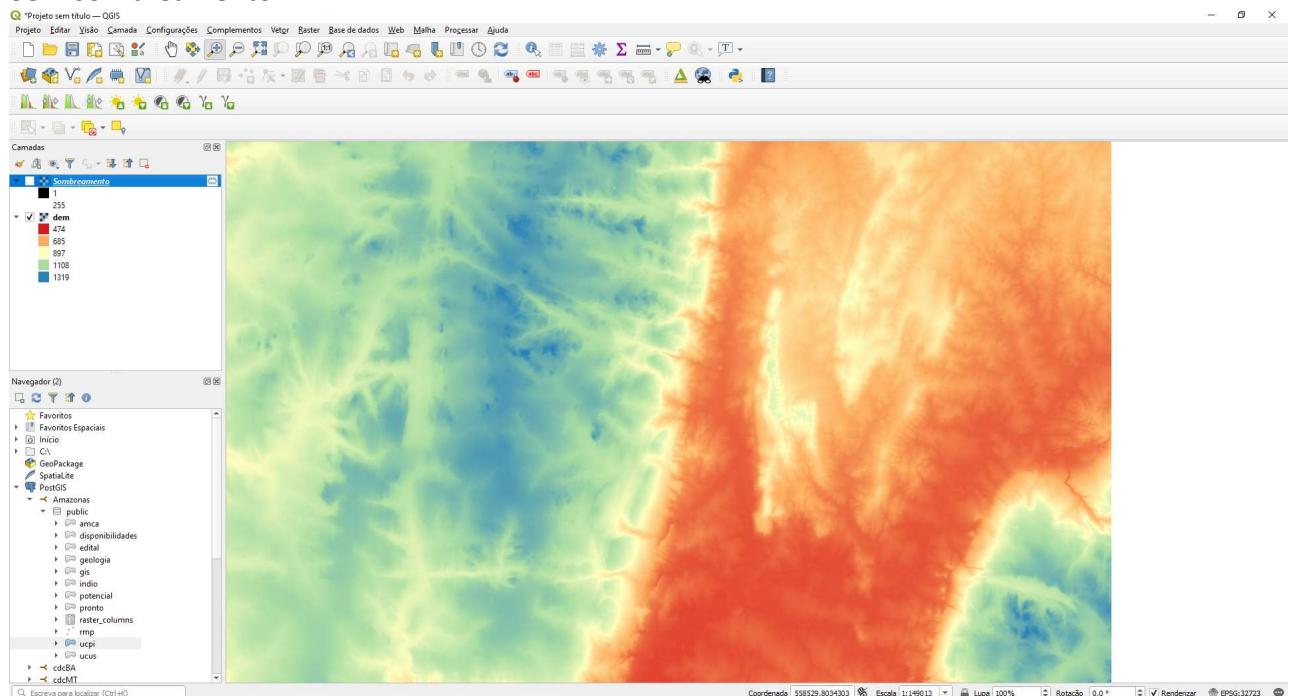
Vamos agora processar o DEM para gerar um sombreamento para realçar o relevo da imagem. No menu selecione **Raster > Análise > Sombreamento** e a janela aparecerá. Clique em **Executar** com os parâmetros apresentados e a imagem Sombreamento será criada.



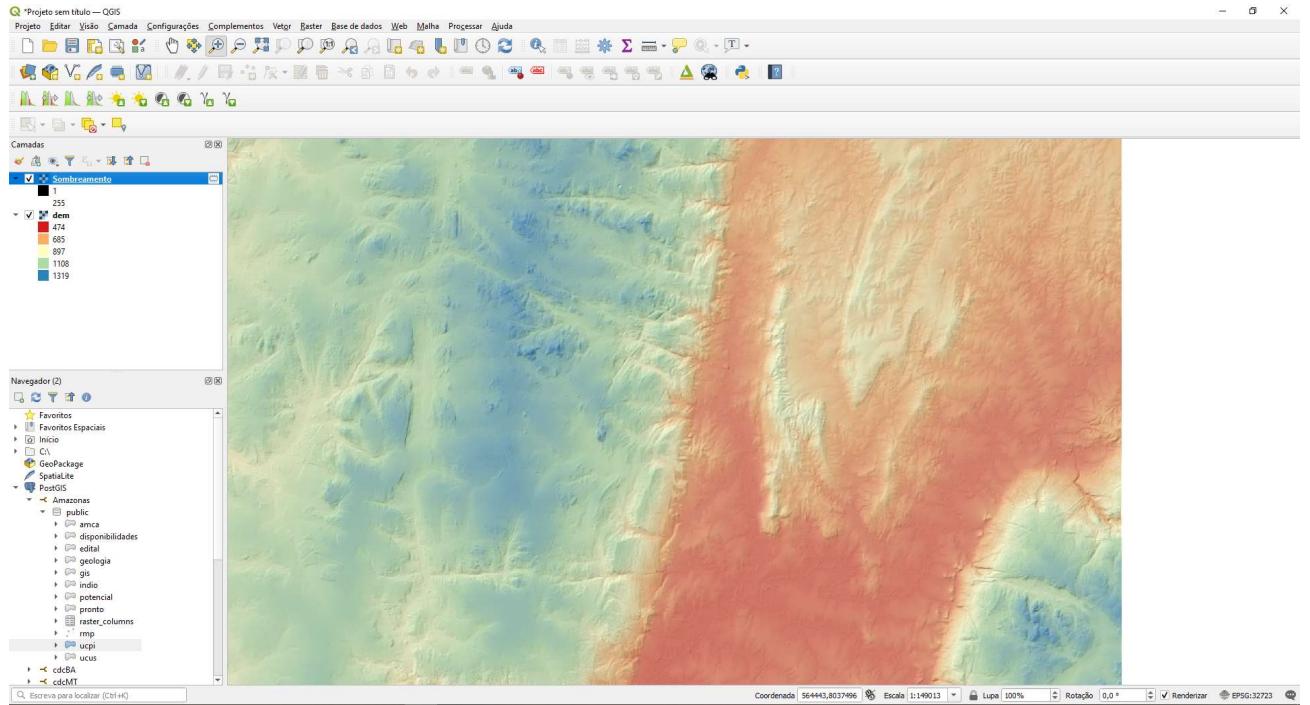


Clique duas vezes em sombreamento e selecione **Transparência** na esquerda, entre com o valor 40% no primeiro campo e clique em **OK**. De um zoom numa região com diferença de relevo e compare o resultado.

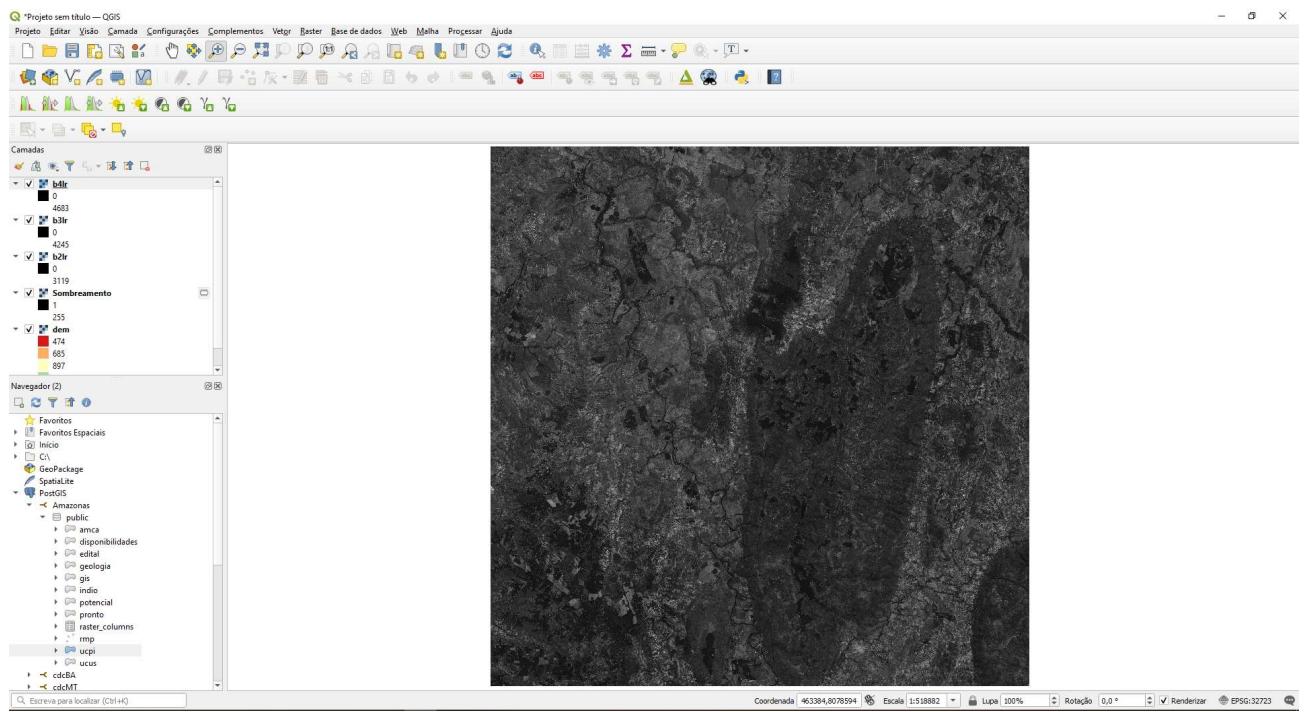
Sem sombreamento:



## Com Sombreamento:

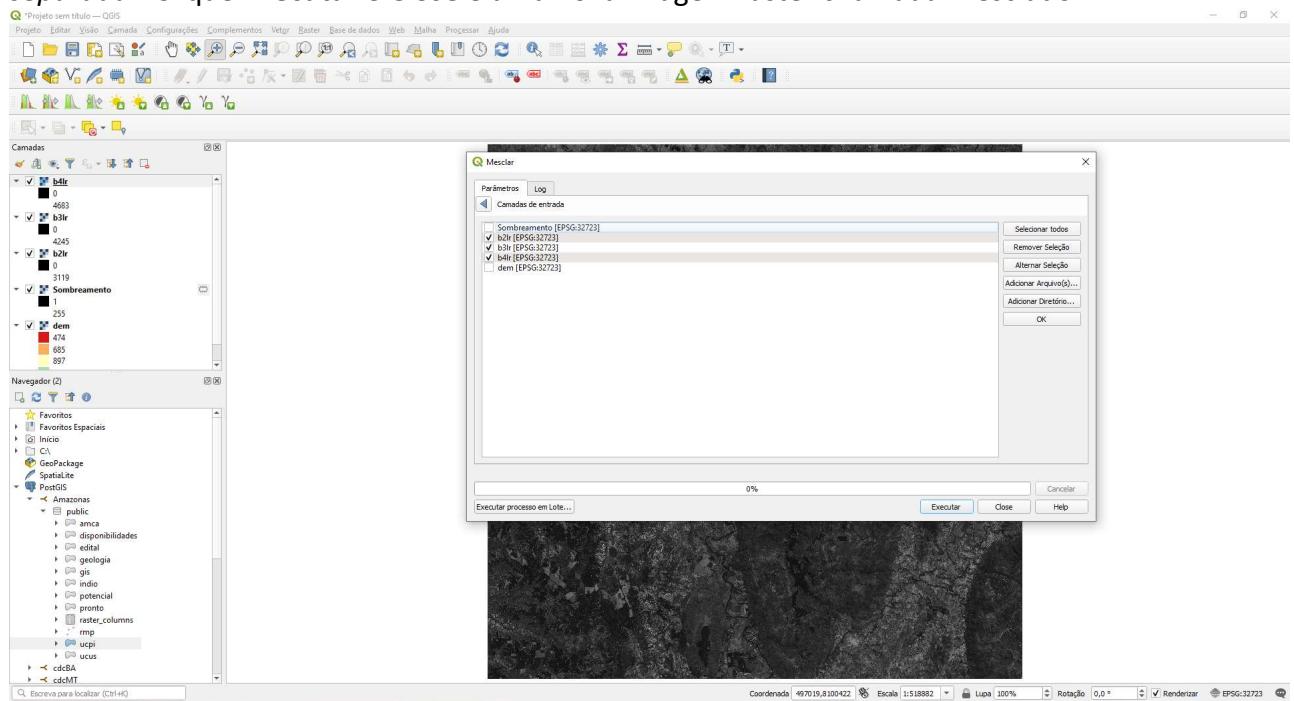


Vamos agora carregar as bandas 2, 3 e 4 do Sentinel2 desta área. Carregue as três bandas (Crtl+Shift+r) e adicione elas ao projeto.

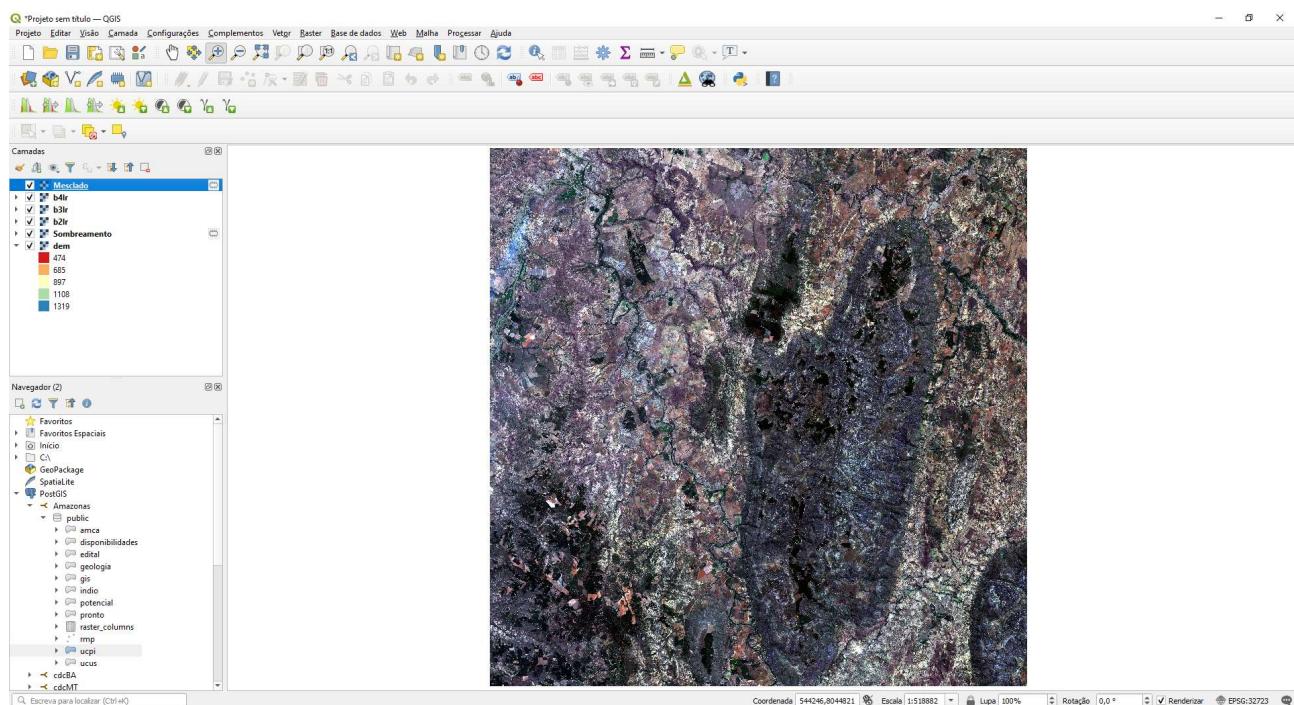


Combinamos estas três bandas para criar uma composição de cor verdadeira (RGB) usando **Raster > Miscelânea > Mesclar**. Selecione as três bandas recém carregadas clicando no ... e marcando elas conforme abaixo.

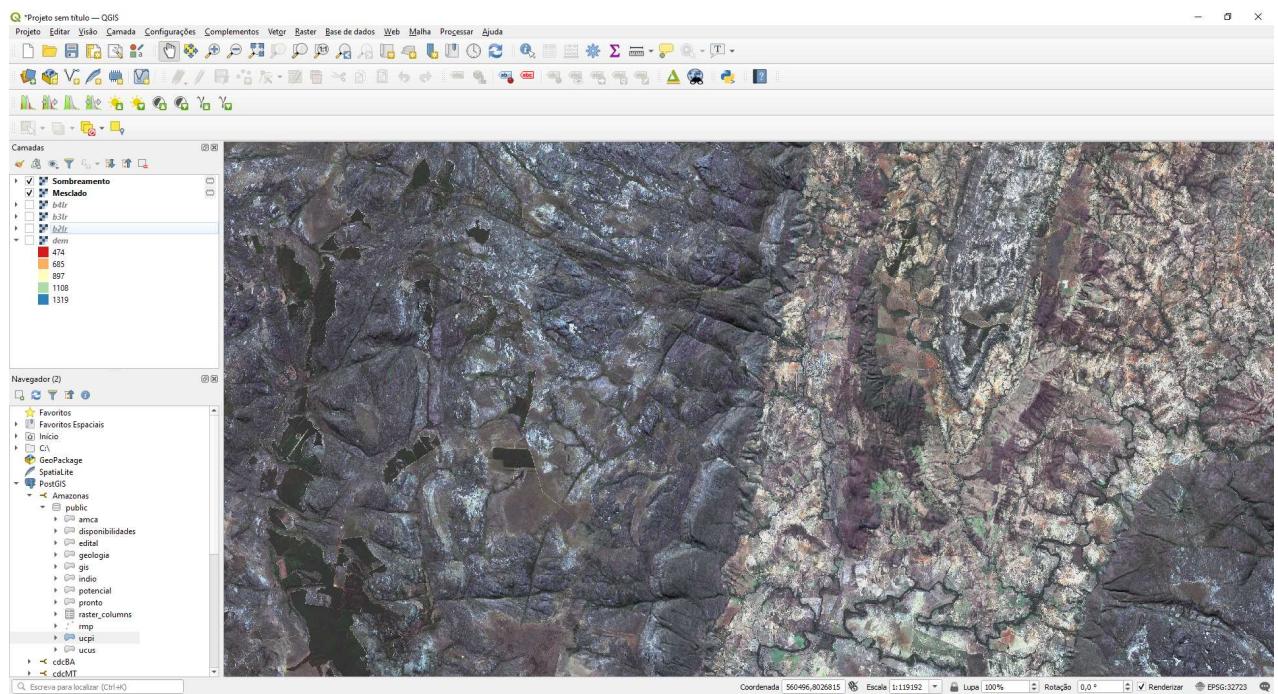
Clique OK na janela de Seleção Múltipla e em seguida coloque cada arquivo de entrada em uma banda separada. Selecione ‘Coloque cada arquivo de entrada em banda separada’. Clique **Executar** e **Close** e uma nova imagem raster chamada Mesclado.



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 Sombreado no painel Camadas, ativar sombreado e dar um zoom numa área. Veremos um resultado assim:

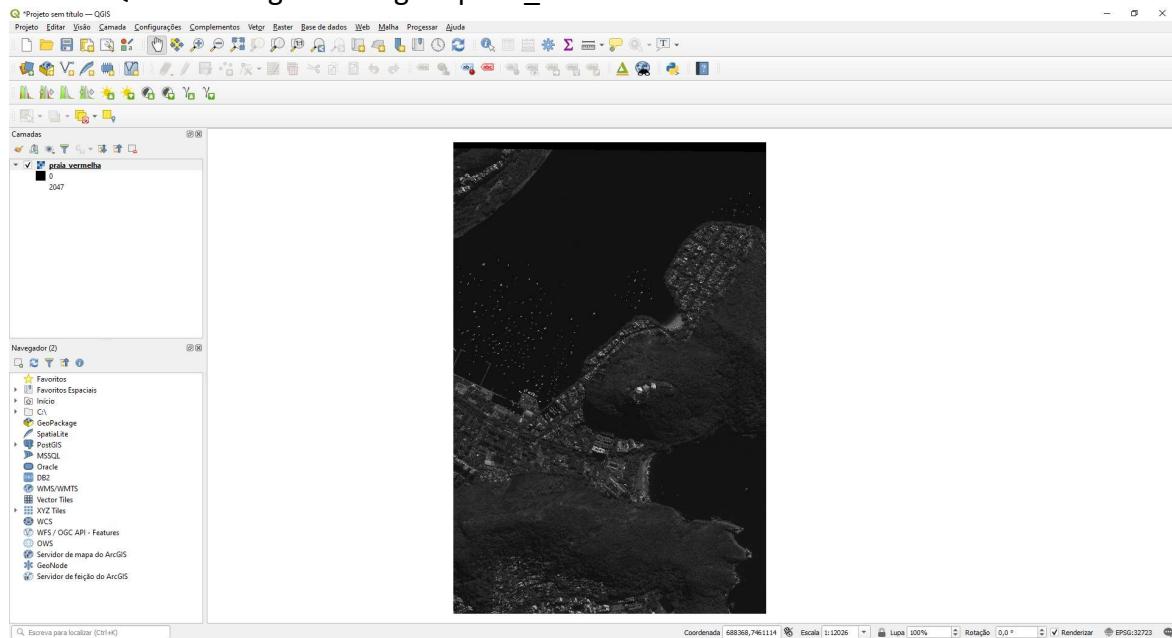


Grave este projeto como **imagens.qgz** para usarmos depois. Muito mais pode ser feito com imagens Raster, essa foi somente uma introdução de como manipular imagens raster. Vamos agora ver com criar vetores e rasters em QGIS.

## 1.4 Criando Dados Vetoriais

Podemos criar 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 camada.

Vamos inicialmente criar uma camada do tipo ponto.

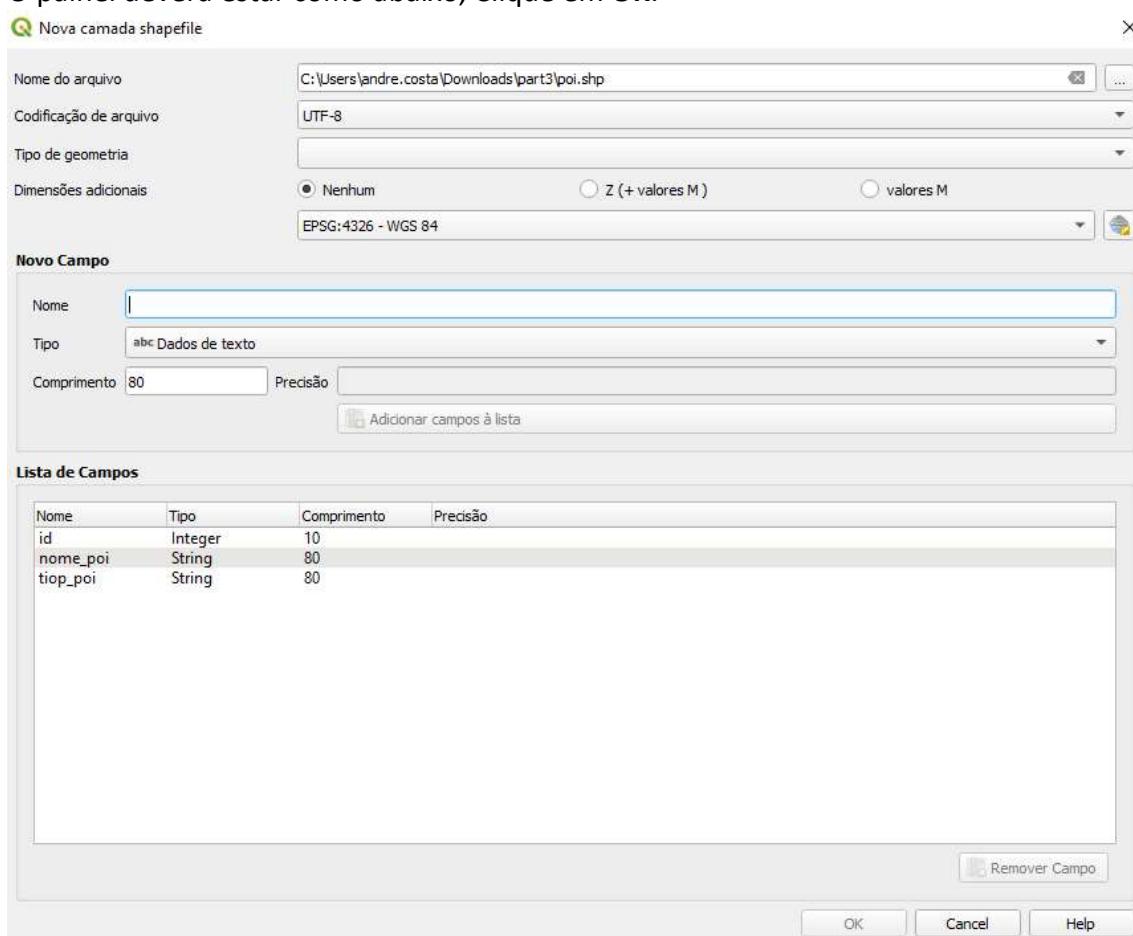
Primeiro criamos o arquivo selecionando o botão ..., navegue até a pasta desejada e crie um arquivo com o nome **poi.shp**

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.

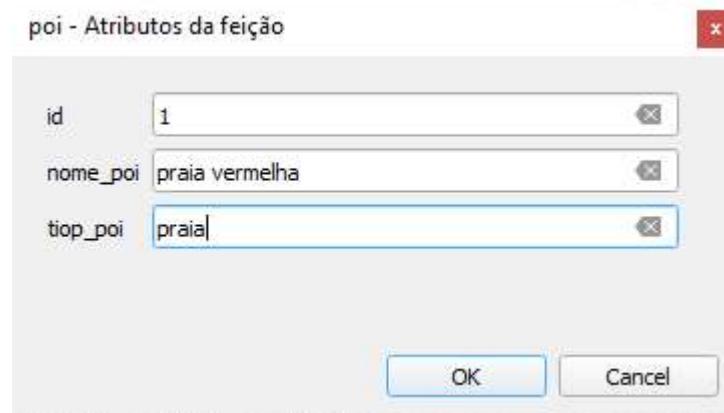
O painel deverá estar como abaixo, Clique em **OK**:



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 pela barra de ferramenta clicando em e depois 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 late Clube.

Atributos com ID =2, nome\_poi= Mariana late 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.

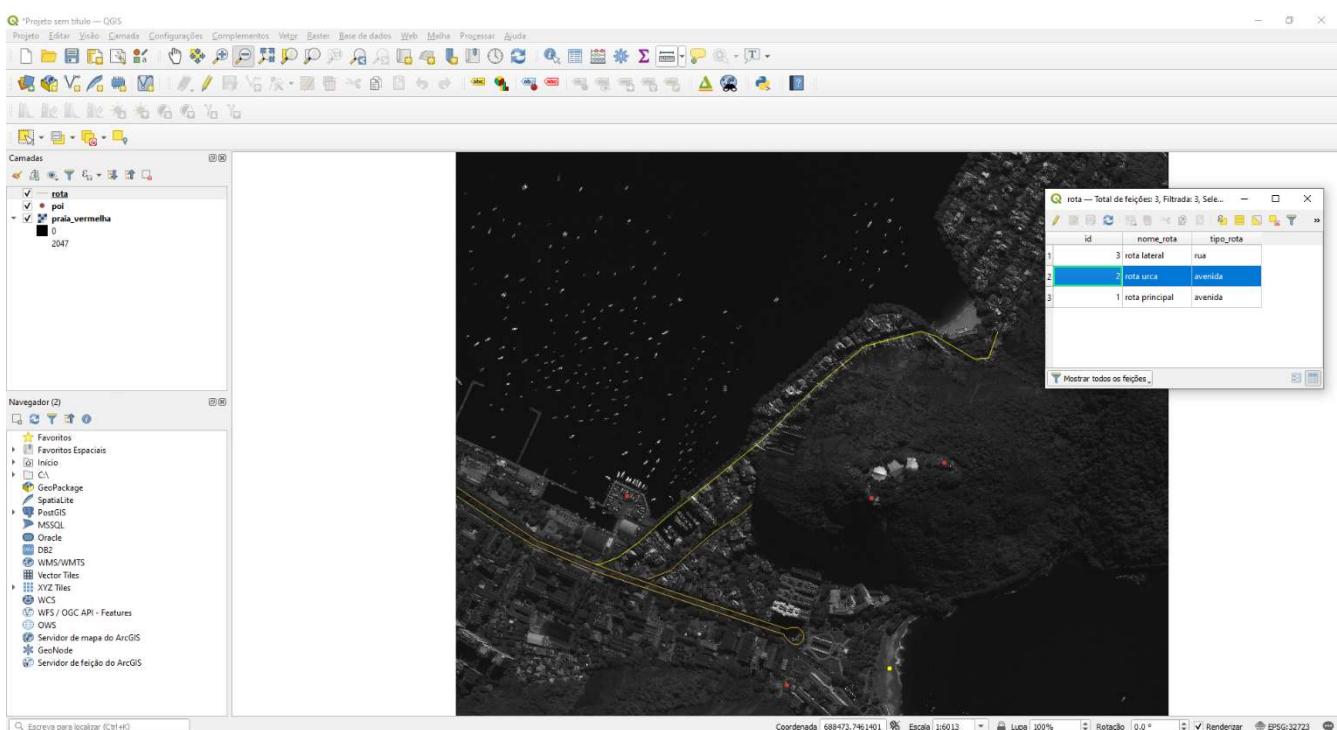
ID	Nome Poi	Tipo Poi
1	3 estação bondin...	estação bondin...
2	Mariana late club	marina
3	praia vermelha	prai
4	5 estação bondin...	estação bondin...
5	4 estação bondin...	estação bondin...

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 camada e crie uma camada do tipo linha com o nome de rota com os atributos nome\_rota e tipo\_rota.

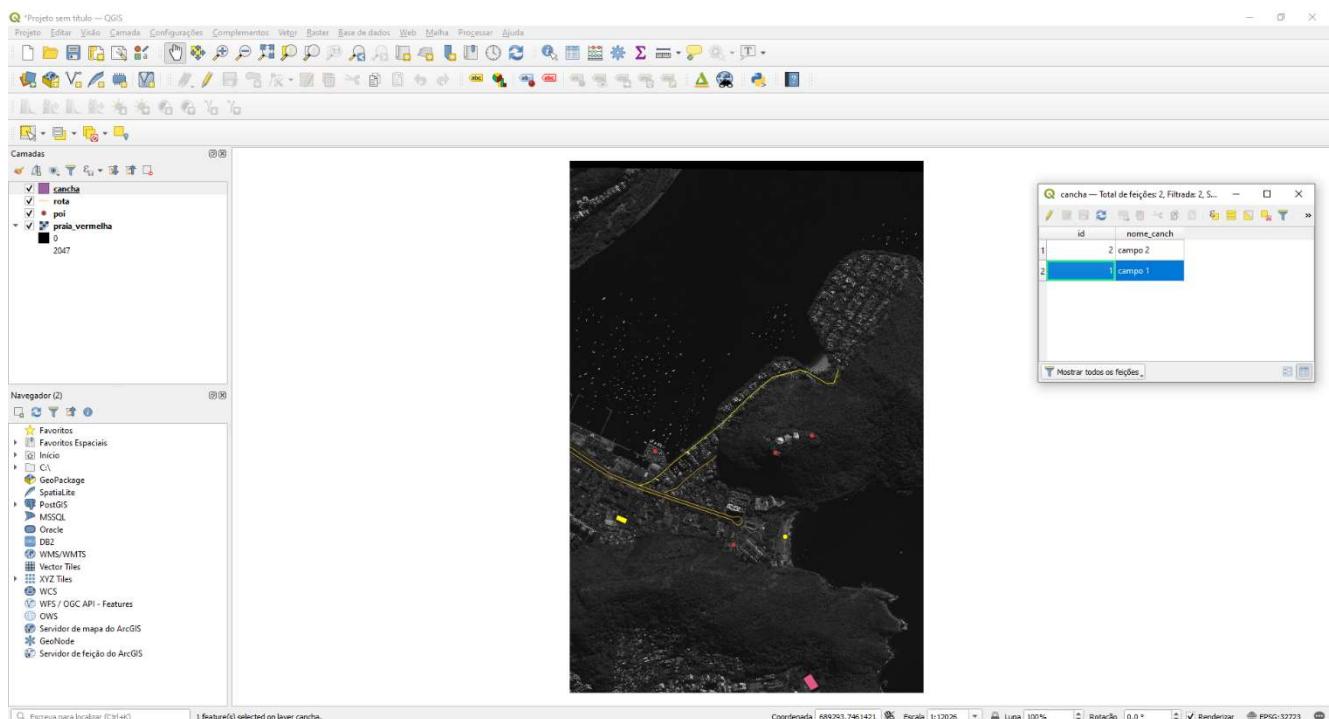
Clique em e digite as linhas clicando com o botão esquerdo do mouse, ao terminar uma avenida clique no botão direito do mouse e entre com os atributos desta linha, repita para digitar duas novas linhas.

Ao final, clique novamente em **Alternar 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.



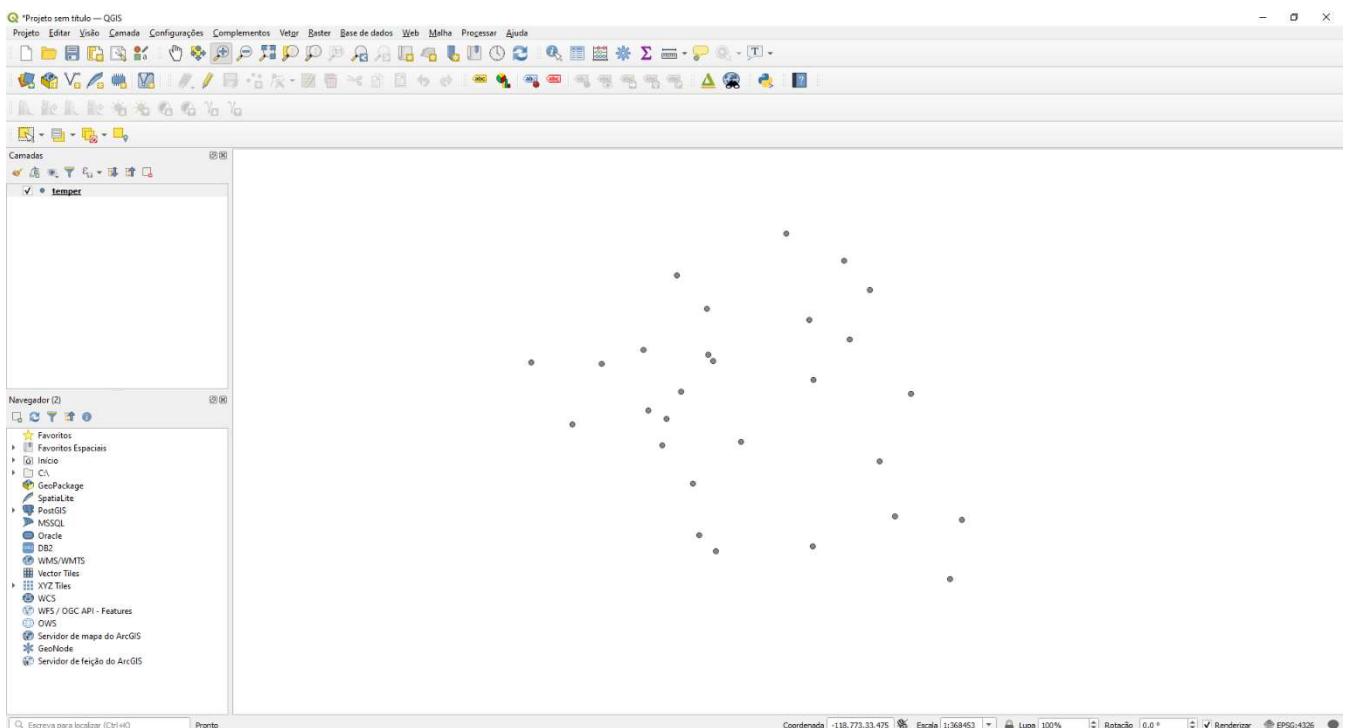
Finalizaremos agora criando uma camada do tipo polígono. Entre em **Camada > Criar Nova Camada > Shapefile** para criar uma camada e crie uma camada do tipo polígono com o nome de cancha com o atributo nome\_canch. Clique em e digite as linhas clicando com o botão esquerdo do mouse, ao terminar um dos campos de futebol clique no botão direito do mouse e entre com os atributos deste polígono, repita para digitar o outro polígono.

Ao final, clique novamente em **Alternar 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.

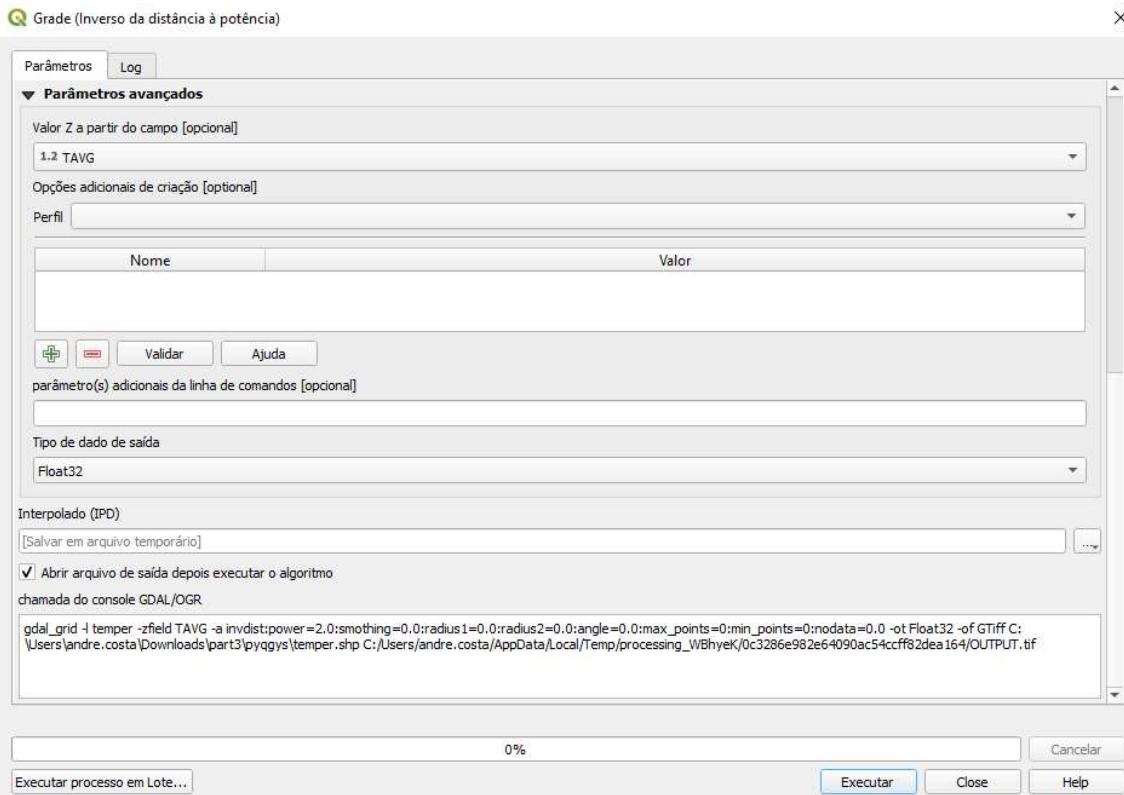


## 1.5 Criando Raster a partir de Pontos Vetoriais

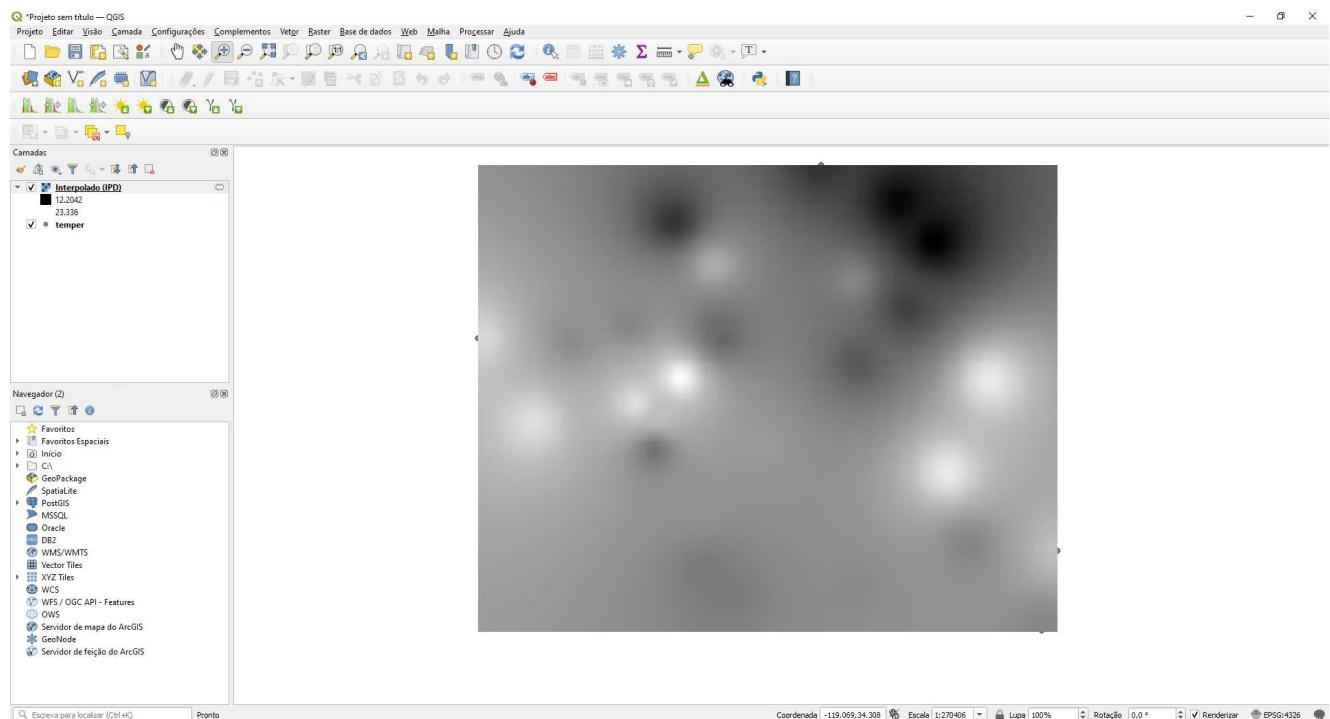
Podemos criar imagens raster a partir de dados pontuais usando interpolação dos dados. Vamos mostrar aqui os passos de como criar este raster. Primeiramente vamos abrir o arquivo **temper.shp** com dados pontuais de temperatura no dia 13 de novembro de 2015 na região de Los Angeles.



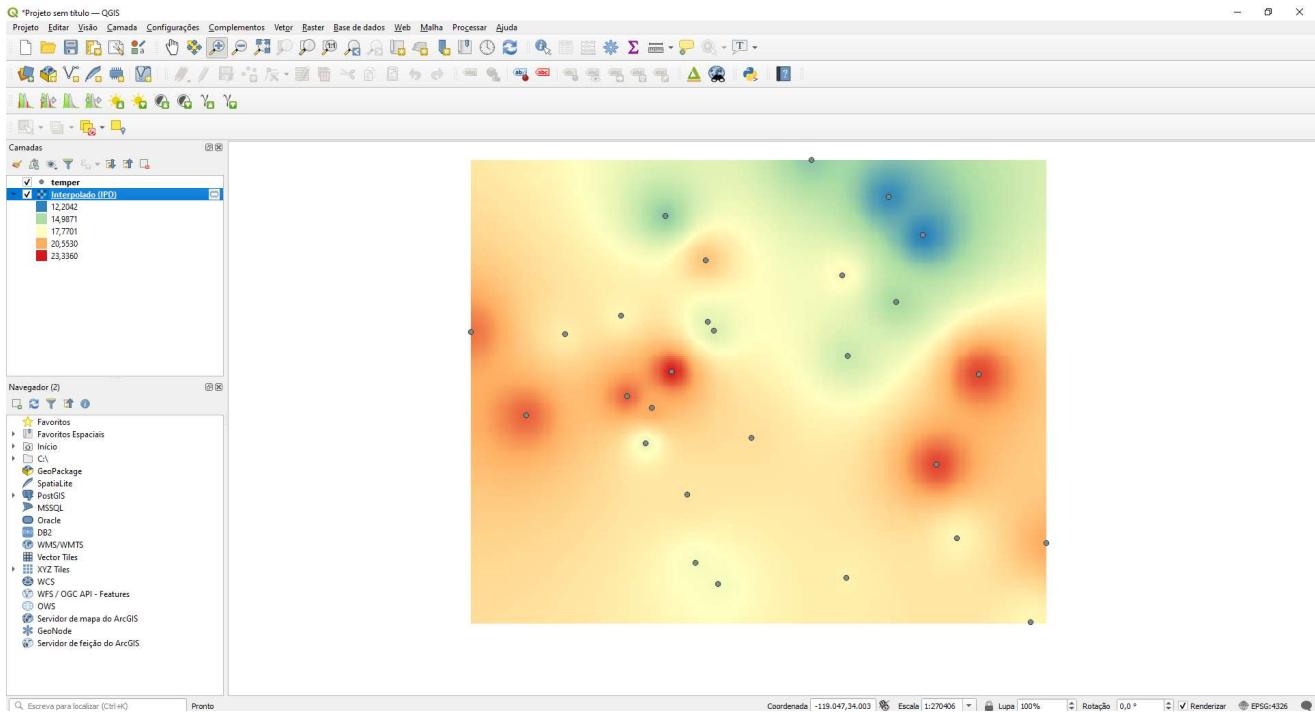
Selecione o menu **Raster > Análise > Grade** (Inverso da distância a Potência). Vamos manter os valores padrões apresentados selecionando somente TAVG no campo Valor Z conforme abaixo.



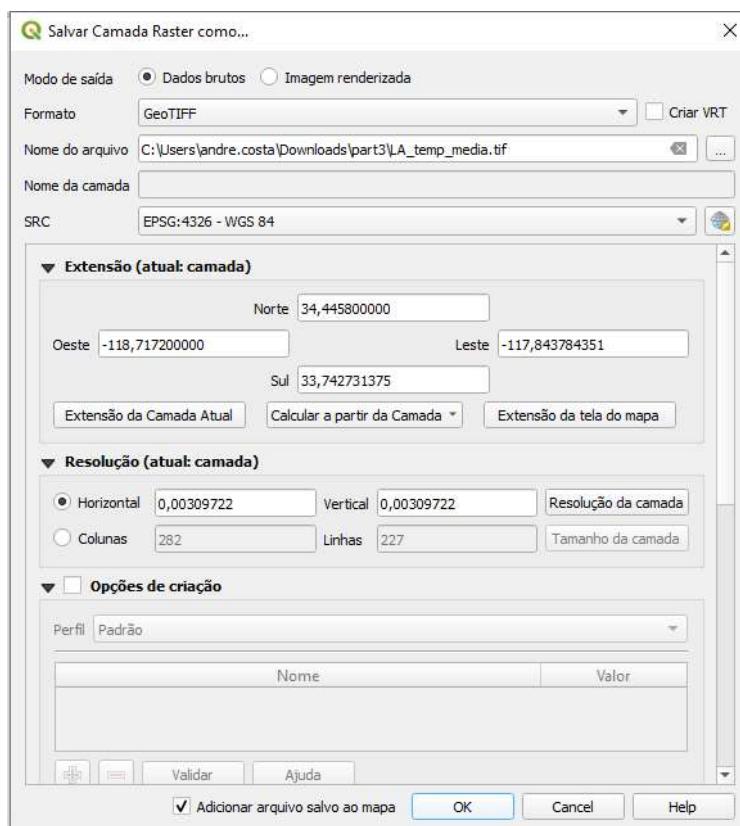
Ao executarmos o seguinte raster aparecerá na tela.



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 as 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 os 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**:

```
Python Console
Use iface to access QGIS API interface or Type help(iface) for more info
Security warning: typing commands from an untrusted source can harm your computer
```

Exemplo ilustrativo de como criar um ponto (veremos a fundo na parte 3 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])
```

Carregando de um banco de dados e gravando como texto em arquivo local

```
>>> tabela = "rmp"
>>> geometria = "geom"
>>> uri = QgsDataSourceUri()
>>> uri.setConnection("pg.amazeone.com.br", "5432", "dnpmam", "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['SUBST_PRIN'],
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 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) é 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]
>>> caipi = ['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]
51
>>> lista[4]
15.99
```

```
>>> 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

```
>>> caipi[3]='pinga'
>>> caipi
['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.

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

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

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

Alguns métodos de interação com listas são mostrados 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]
('banana', 3, 45, False, 'oi!')
>>> doist[0][0]
'banana'
>>> doist[1][0]
1
>>> t[0]='maçã'
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>
TypeError: 'tuple' object does not support item assignment
```

### 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
>>> 'mar' in conjunto
False
>>> num ={1,2,3,2,3,2,1,23,'a'}
>>> num
{1, 2, 3, 'a', 23}
>>> cidade=set('pindamonhangaba')
>>> cidade
{'p', 'a', 'h', 'm', 'g', 'n', 'i', 'o', 'b', 'd'}\
```

### 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)
'\u0311'
>>> ord('\u0311')
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

```

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

```
and      except      lambda      with
as       finally     nonlocal    while
assert   False       None        yield
break    for         not
class    from        or
continue global      pass
def      if          raise
del      import     return
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 recuo de script.

Obrigatoriamente em python temos de usar recuo 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 recuo 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=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. Nossa 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 a força da linguagem python com inúmeros módulos existentes para as mais diversas funções. Vamos iniciar vendo o módulo pandas.

## 2.5 Pandas

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

Panda tira vantagem em utilizar outra biblioteca 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: []

```

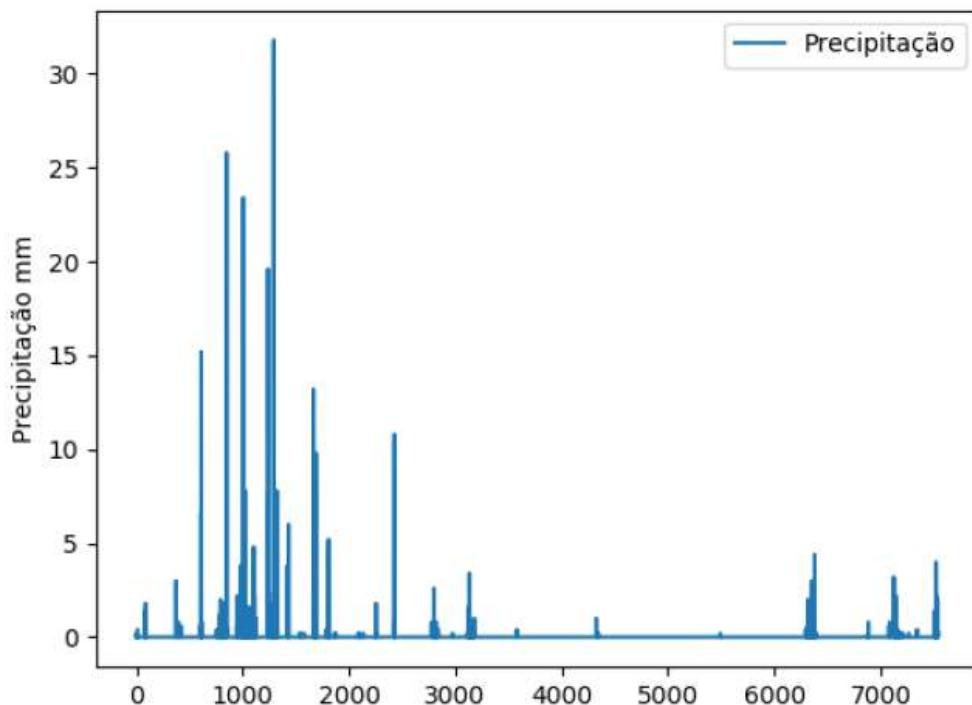
## 2.6 Gráficos

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

```

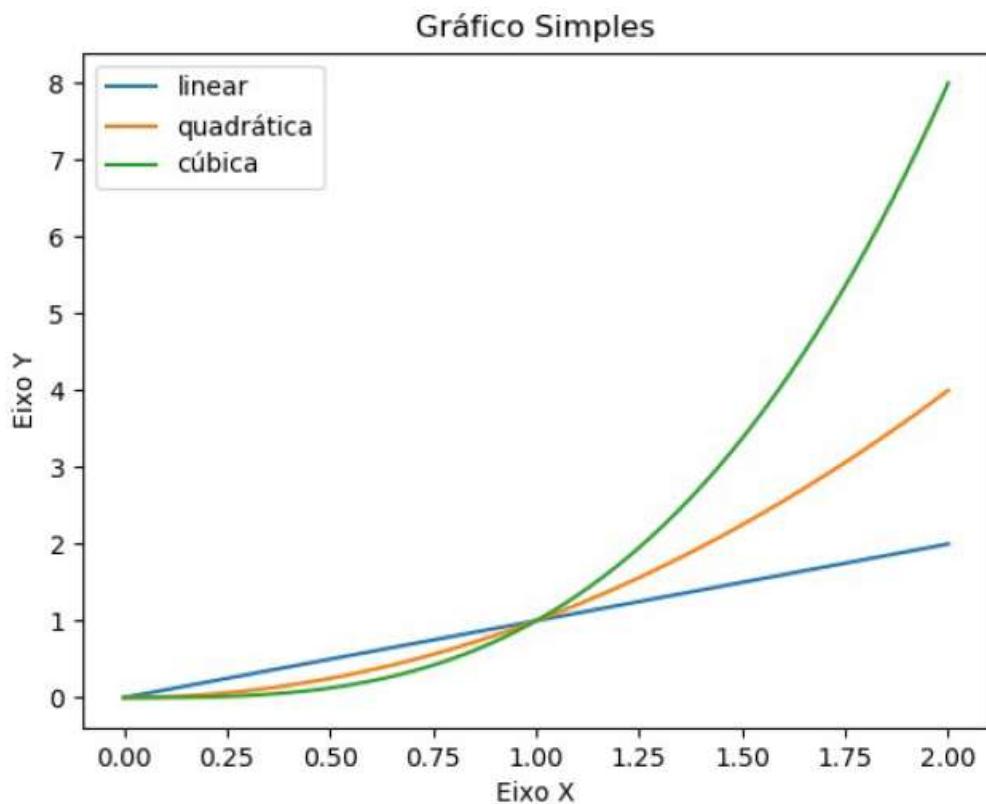
>>> import pandas as pd
>>> import matplotlib.pyplot as plt
>>> data = pd.read_csv('curvelo.csv')
>>> plt.plot(data[['precipitacao']], label='Precipitação')
[<matplotlib.lines.Line2D object at 0x7f55bbc70d90>]
>>> plt.ylabel('Precipitação mm')
Text(0, 0.5, 'Precipitação mm')
>>> plt.legend()
<matplotlib.legend.Legend 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. Usando 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 biblioteca PyQGIS é bastante extensa com diversas classes. Vamos aqui cobrir as classes mais básicas e essenciais para a partir deste ponto termos 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 objeto usando o método `QgsProject.instance()`. Vamos mostrar como criar um projeto vazio chamado **meu\_projeto.qgs** usando o método `write()`.

```
>>> projeto= QgsProject.instance()
>>> # ajuste caminho para o arquivo a ser criado com o seu sistema
>>> 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
```

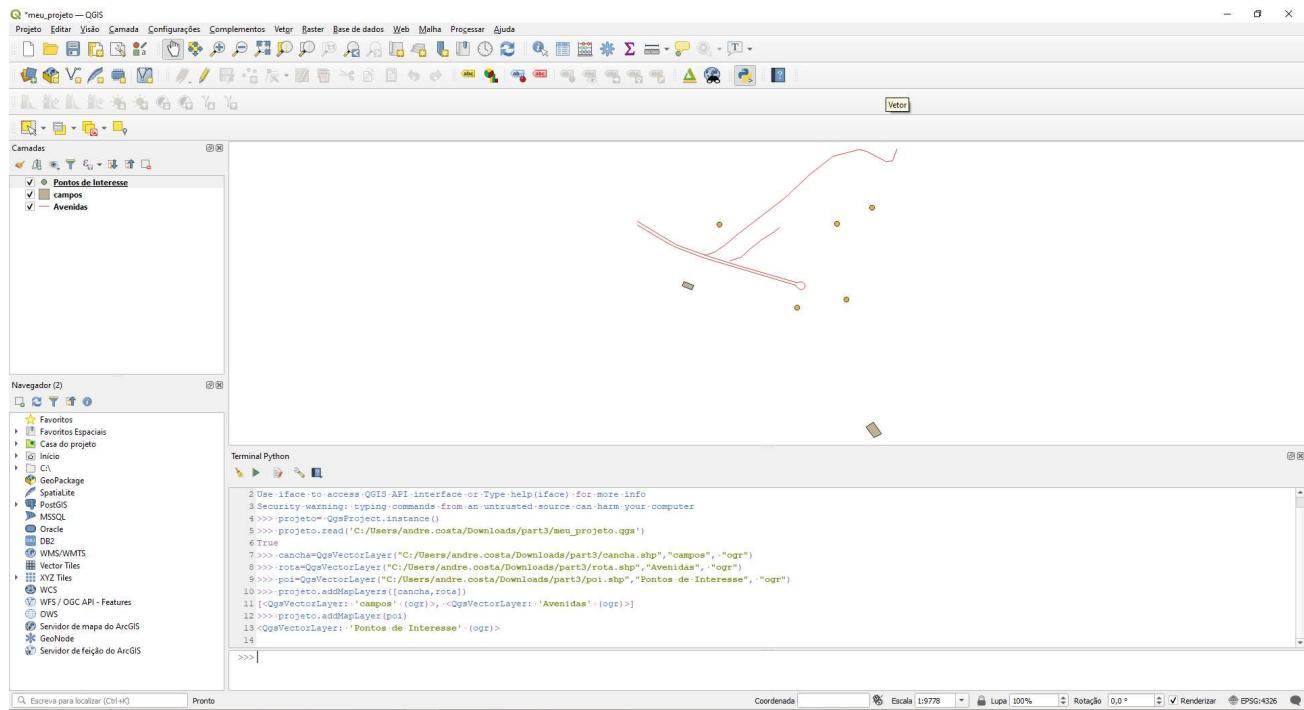
Na medida que formos vendo as outras classes, vamos ver outros métodos associados à classe Projeto.

##### [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()
```

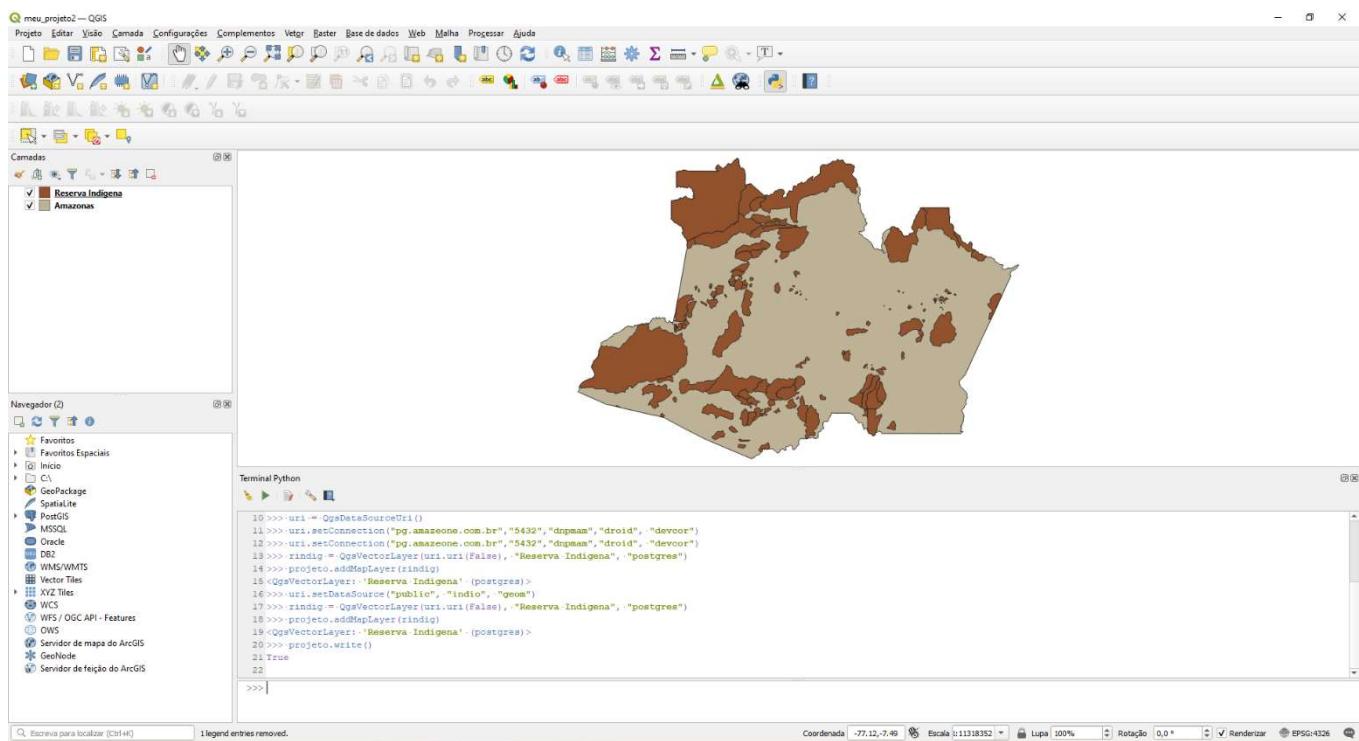
Algo similar ao apresentado abaixo deverá aparecer:



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()
>>> #ajustar caminho dos arquivos de acordo com seu sistema
>>> projeto.write('c:/users/voce/meu_projeto2.qgs')
>>> am=QgsVectorLayer("c:/users/voce/amazonas.shp", "AM", "ogr")
>>> projeto.addMapLayer(am)
>>> uri = QgsDataSourceUri()
>>> uri.setConnection("pg.amazeone.com.br", "5432", "dnpmam", "droid",
"devcor")
>>> uri.setDataSource("public", "indio", "geom")
>>> rindig = QgsVectorLayer(uri.uri(False), "Reserva Indígena",
"postgres")
>>> projeto.addMapLayer(rindig)
>>> projeto.write()
```

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. Um projeto conforme o ilustrado abaixo deverá aparecer.

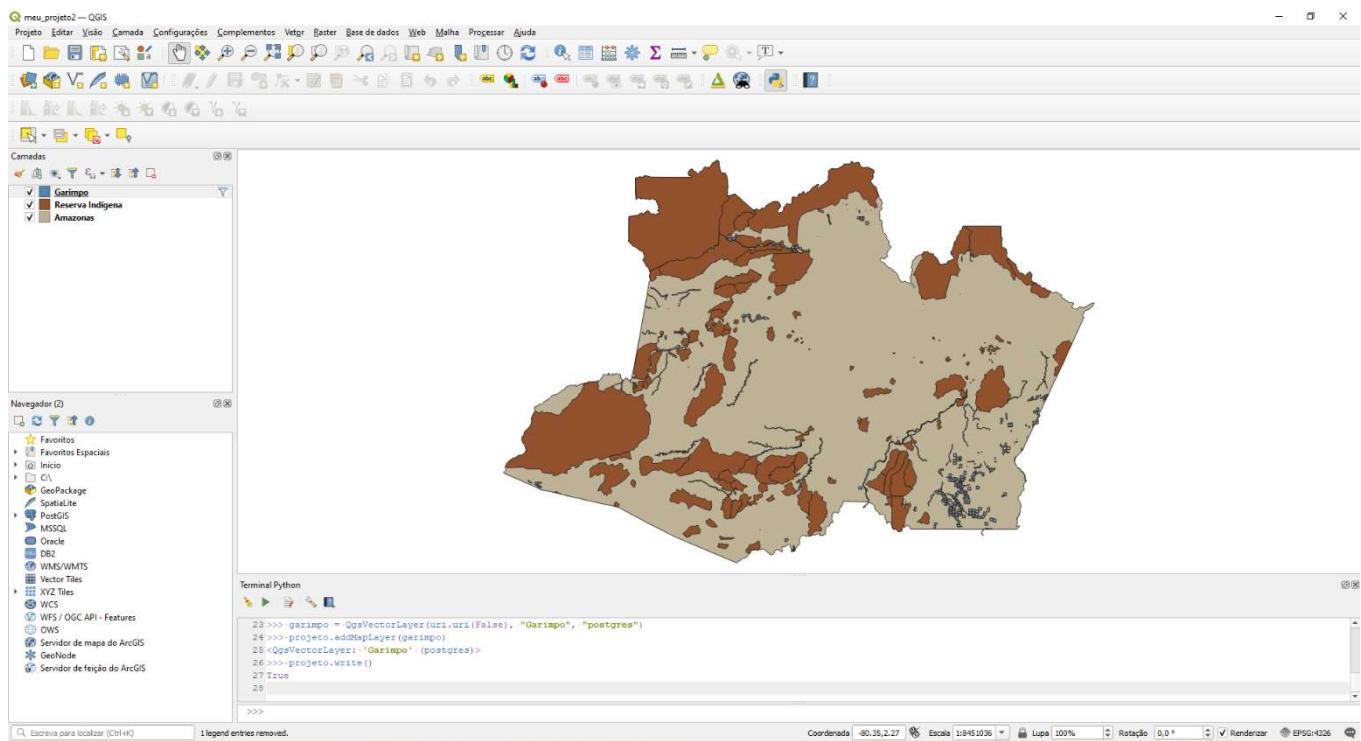


Alternativamente podemos adicionar uma cláusula SQL WHERE como o quarto argumento. Vamos ver um exemplo onde extraímos uma camada vetorial somente os requerimentos de garimpo e permissão de lavra garimpeira dos requerimentos do estado do Amazonas e adicionamos ela no nosso projeto já criado acima.

```

>>> 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 (QgsProject) relacionados a classe Camadas Vetoriais (QgsVectorLayer).

count retorna o número de camadas válidas do projeto.

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

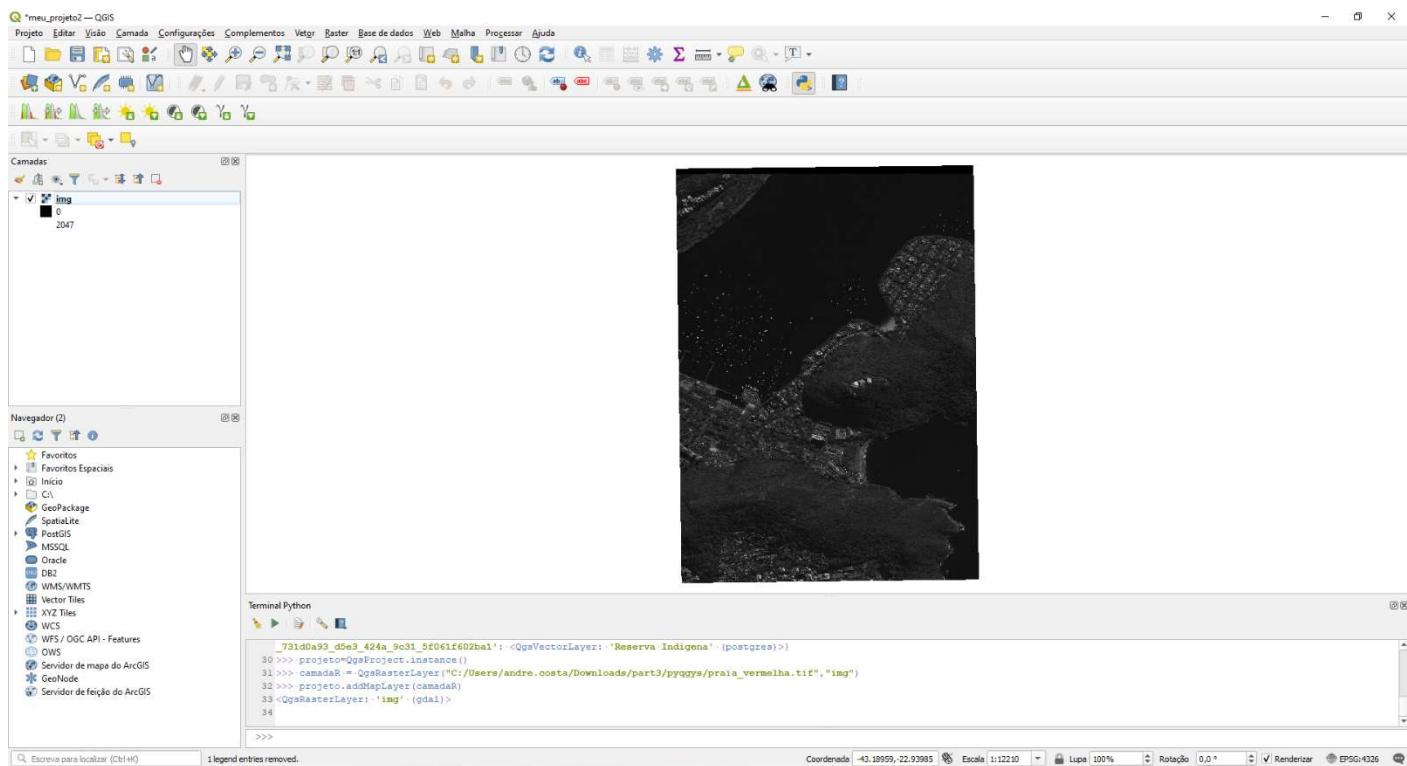
mapLayers retorna um mapa das camadas existentes do projeto.

```
>>> projeto.mapLayers()  
{'Amazonas_143625b7_7552_40a1_9569_b7e0de0ba3a7': <QgsVectorLayer:  
'Amazonas' (ogr)>, 'Garimpo_6af241f6_9158_4349_9cb6_8aac3cc3a37d':  
<QgsVectorLayer: 'Garimpo' (postgres)>,  
'Reserva_Indígena_731d0a93_d5e3_424a_9c31_5f061f602ba1':  
<QgsVectorLayer: 'Reserva Indígena' (postgres)>}
```

### Classe Raster (QgsRasterLayer) – Adicionar Camada Raster

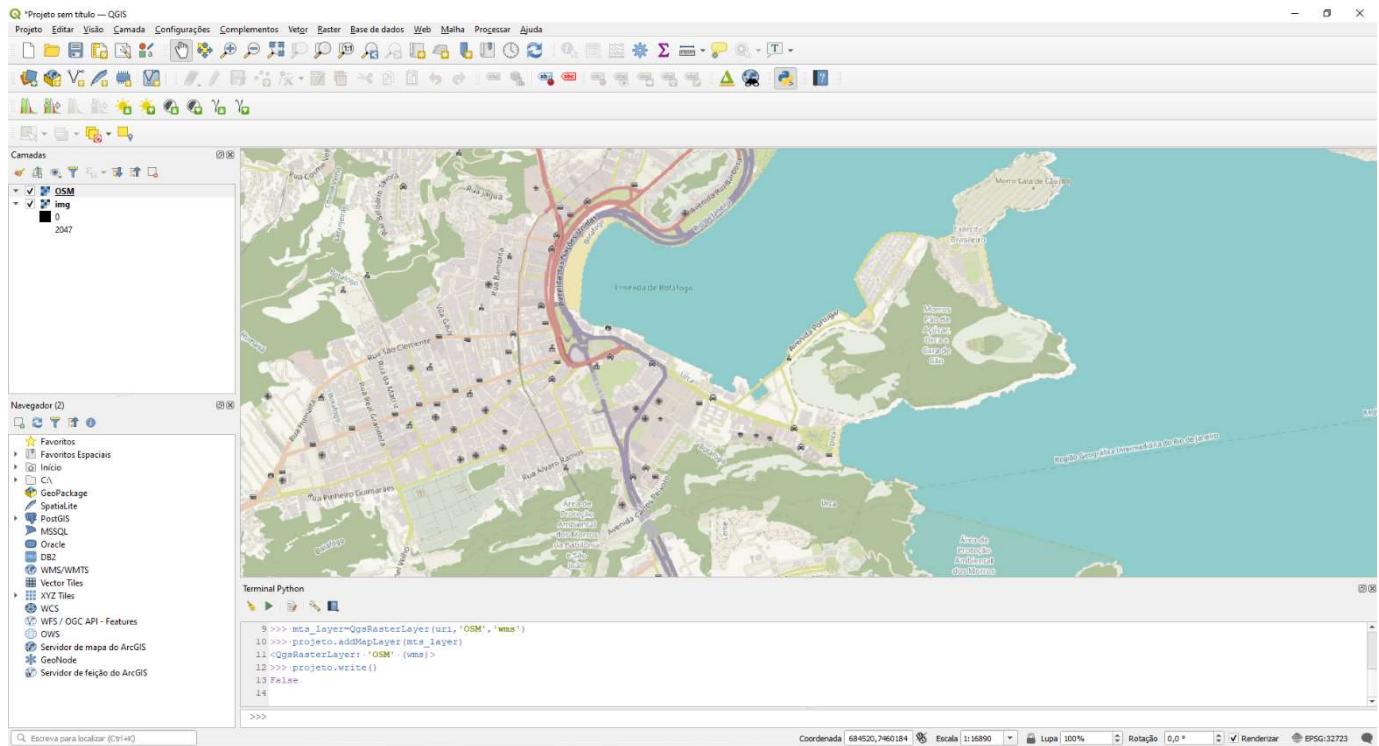
Similar à forma que adicionamos camadas vetoriais, podemos adicionar imagens raster no nosso projeto usando objeto da classe raster. Vamos criar um projeto e adicionar uma imagem raster nele.

```
>>> projeto=QgsProject.instance()  
>>> camadaR = QgsRasterLayer("c:/users/voce/praiavermelha.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 (WebMapaService).

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



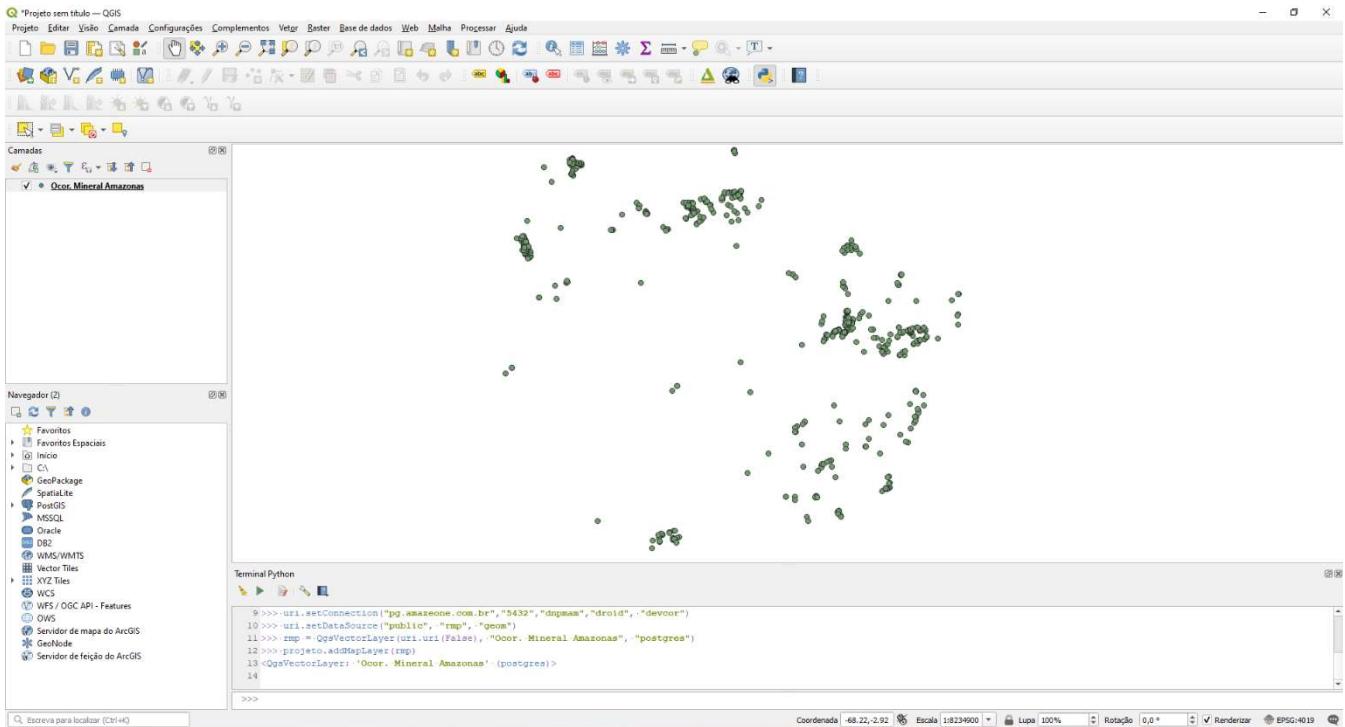
### 3.2 Interagindo com informações de objetos da classe Vector

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("pg.amazeone.com.br", "5432", "dnpmam", "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 mais usadas de maneira geral. Outras informações existem no objeto camada, veja a documentação para mais detalhes.



## 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())
Unknown datum based upon the GRS 1980 ellipsoid
```

## 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)
-70.1 -9.53860000030878 -56.7497 2.21390000007294
```

## 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: 624
```

## 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())
codigo_obj text
TOPONIMIA text
latitude float8
longitude float8
SUBST_PRIN text
subst_sec text
abrev text
STATUS_ECO text
grau_de_im text
metodo_geo text
erro_metod text
data_cad text
classe_utí text
tipologia text
classe_gen text
modelo_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
municipio text
uf 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='dnpmam'
host=pg.amazeone.com.br port=5432 user='droid' key='tid'
checkPrimaryKeyUnicity='1' table="public"."rmp" (geom)</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></td></tr>
<tr><td class="highlight">Geometria</td><td>Point (Point)</td></tr>
<tr><td class="highlight">SRC</td><td>EPSG:4019 - Unknown datum based
upon the GRS 1980 ellipsoid - Geográfico</td></tr>
<tr><td class="highlight">Extensão</td><td>-70.099999999999943,-
9.5386000003087794 : -56.749699999999971,2.2139000000729401</td></tr>
<tr><td class="highlight">Unidade</td><td>graus</td></tr>
<tr><td class="highlight">Contagem de feições</td><td>624</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>
</table>
<br><br>
<h1>Extensão</h1>
<hr>
<table class="list-view">
<tr><td class="highlight">CRS</td><td>EPSG:4019 - Unknown datum based
upon the GRS 1980 ellipsoid - 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>26</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>text</td><td>-
1</td><td>0</td><td></td></tr>
<tr class="odd-row"><td>TOPONIMIA</td><td>text</td><td>-
1</td><td>0</td><td></td></tr>
<tr ><td>latitude</td><td>float8</td><td>-
1</td><td>0</td><td></td></tr>
<tr class="odd-row"><td>longitude</td><td>float8</td><td>-
1</td><td>0</td><td></td></tr>
<tr ><td>SUBST_PRIN</td><td>text</td><td>-
1</td><td>0</td><td></td></tr>
<tr class="odd-row"><td>subst_sec</td><td>text</td><td>-
1</td><td>0</td><td></td></tr>
<tr ><td>abbrev</td><td>text</td><td>-1</td><td>0</td><td></td></tr>
<tr class="odd-row"><td>STATUS_ECO</td><td>text</td><td>-
1</td><td>0</td><td></td></tr>
<tr ><td>grau_de_im</td><td>text</td><td>-
1</td><td>0</td><td></td></tr>
<tr class="odd-row"><td>metodo_geo</td><td>text</td><td>-
1</td><td>0</td><td></td></tr>
<tr ><td>erro_metod</td><td>text</td><td>-
1</td><td>0</td><td></td></tr>
<tr class="odd-row"><td>data_cad</td><td>text</td><td>-
1</td><td>0</td><td></td></tr>

```

```

<tr ><td>classe_uti</td><td>text</td><td>-
1</td><td>0</td><td></td></tr>
<tr class="odd-row"><td>tipologia</td><td>text</td><td>-
1</td><td>0</td><td></td></tr>
<tr ><td>classe_gen</td><td>text</td><td>-
1</td><td>0</td><td></td></tr>
<tr class="odd-row"><td>modelo_dep</td><td>text</td><td>-
1</td><td>0</td><td></td></tr>
<tr ><td>assoc_geoq</td><td>text</td><td>-
1</td><td>0</td><td></td></tr>
<tr class="odd-row"><td>rocha_enca</td><td>text</td><td>-
1</td><td>0</td><td></td></tr>
<tr ><td>rocha_hosp</td><td>text</td><td>-
1</td><td>0</td><td></td></tr>
<tr class="odd-row"><td>textura_mi</td><td>text</td><td>-
1</td><td>0</td><td></td></tr>
<tr ><td>tipos_alte</td><td>text</td><td>-
1</td><td>0</td><td></td></tr>
<tr class="odd-row"><td>extrmin_x_</td><td>text</td><td>-
1</td><td>0</td><td></td></tr>
<tr ><td>assoc_mine</td><td>text</td><td>-
1</td><td>0</td><td></td></tr>
<tr class="odd-row"><td>origem</td><td>text</td><td>-
1</td><td>0</td><td></td></tr>
<tr ><td>municipio</td><td>text</td><td>-
1</td><td>0</td><td></td></tr>
<tr class="odd-row"><td>uf</td><td>text</td><td>-
1</td><td>0</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 num navegador assim:**

Informação do provedor

---

Nome	Ocor. Mineral Amazonas
Fonte	dbname='dnpmam' host=pg.amazeone.com.br port=5432 user='droid' key='tid' checkPrimaryKeyUnicity='1' table="public"."rmp" (geom)
Armazenamento	PostgreSQL database with PostGIS extension
Comentário	

## Codificação

Geometria	Point (Point)
SRC	EPSG:4019 - Unknown datum based upon the GRS 1980 ellipsoid – Geográfico
Extensão	-70.099999999999943,-9.5386000003087794 : -56.749699999999971,2.2139000000729401
Unidade	Graus
Contagem de feições	624
Identificação	

---

## Identifier

### Parent Identifier

### Title

Type dataset

### Language

### Abstract

### Categories

### Keywords

### Extensão

CRS EPSG:4019 - Unknown datum based upon the GRS 1980 ellipsoid – Geographic

### Spatial Extent

### Temporal Extent

### Acesso

### Fees

### Licenses

Rights

Constraints

Campos

---

Contagem 26

Campo	Tipo	Comprimento	Precisão	Comentário
codigo_obj	text	-1	0	
TOPONIMIA	text	-1	0	
Latitude	float8	-1	0	
Longitude	float8	-1	0	
SUBST_PRIN	text	-1	0	
subst_sec	text	-1	0	
Abrev	text	-1	0	
STATUS_ECO	text	-1	0	
grau_de_im	text	-1	0	
metodo_geo	text	-1	0	
erro_metod	text	-1	0	
data_cad	text	-1	0	
classe_uti	text	-1	0	
Tipologia	text	-1	0	
classe_gen	text	-1	0	
modelo_dep	text	-1	0	
assoc_geoq	text	-1	0	
rocha_enca	text	-1	0	
rocha_hosp	text	-1	0	
textura_mi	text	-1	0	

tipos_alte	text	-1	0
extrmin_x_	text	-1	0
assoc_mine	text	-1	0
Origem	text	-1	0
Município	text	-1	0
Uf	text	-1	0

## 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.

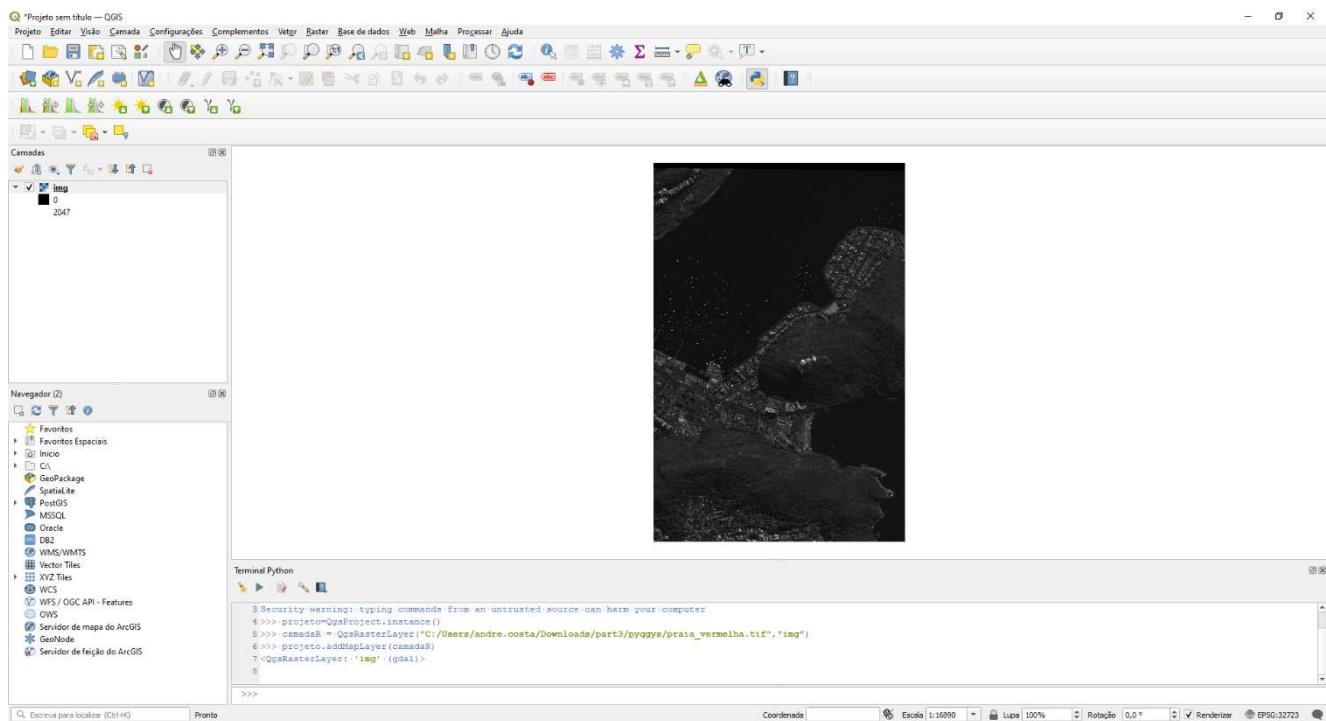
```
>>> elementos=rmp.getFeatures()
>>> for e in elementos:
...     attr=e.attributes()
...     print (attr)
[('25368', 'APUI', -7.7972, -58.8558, 'Calcário', NULL, 'cc', 'Não
explorado', 'Depósito', 'Levantamento em Carta 1:250.000', '250 a
1.000 m', '2001/11/26', 'Insumos para agricultura', NULL, NULL, NULL,
NULL, NULL, NULL, NULL, NULL, '7', 'APUI', 'AM')
...
...
[('46734', 'RIO MANICORÉ', -6.11, -61.5669, 'Argila', NULL, 'arg',
'(Não determinado)', 'Depósito', 'GPS Manual pré 25/05/2000', '50 a
200 m', '2006/11/24', 'Material de uso na construção civil', NULL,
NULL, NULL, NULL, NULL, NULL, NULL, NULL, '2', 'MANICORE',
'AM')]
```

### 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
0.299999999999144

>>> camadaR.rasterUnitsPerPixelY() #resolução em Y
0.299999999999545
```

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 o código abaixo em vez de usar o método `rasterUnitsPerPixelX()`:

```
>>> (camadaR.extent().xMaximum() -  
camadaR.extent().xMinimum()) / camadaR.width()  
0.29999999999999144
```

Podemos também visualizar informações usando o método `htmlMetadata()`.

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

Informação do provedor

---

Nome	Img
Caminho	<a href="C:\Users\andre.costa\Downloads\part3\pyqgys\praia_vermelha.tif">C:\Users\andre.costa\Downloads\part3\pyqgys\praia_vermelha.tif</a>
SRC	EPSG:32723 - WGS 84 / UTM zone 23S - Projetado
Extensão	687173.99999999835847,7459532.4000000003725290 : 688805.699999998370185,7461990.00000000000000000000
Unidade	Metros
Largura	5439
Altura	8192
tipo de dado	UInt16 - Inteiro de 16 bits sem sinal
Descrição do driver GDAL	GTiff
Metadados do driver	GeoTIFF
GDAL	
Descrição do registro	C:/Users/andre.costa/Downloads/part3/pyqgys/praiavermelha.tif
Compressão	
	STATISTICS_APPROXIMATE=YES
	STATISTICS_MAXIMUM=2047
	STATISTICS_MEAN=220.86768822394
Banda 1	STATISTICS_MINIMUM=0
	STATISTICS_STDDEV=156.88305888003
	STATISTICS_VALID_PERCENT=100

AREA\_OR\_POINT=Area

TIFFTAG\_COPYRIGHT=(C) COPYRIGHT 2016 DigitalGlobe, Inc., Longmont CO  
USA 80503

Mais informações

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 0.299999999999999889,-0.299999999999999889

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 objeto 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)
```

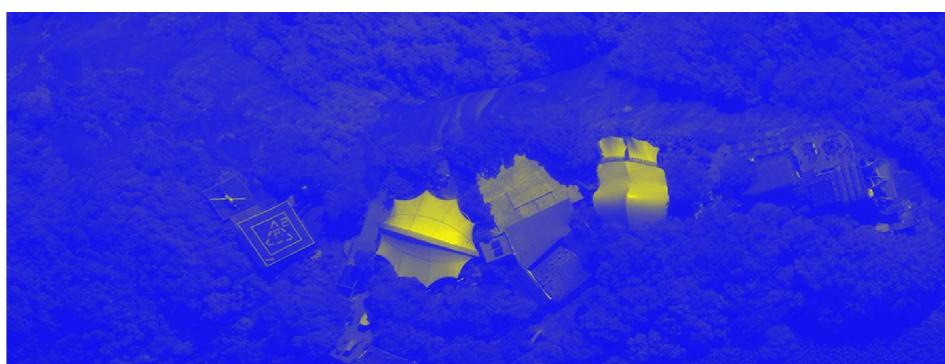
Finalmente criamos o objeto renderizador de cor com: dados do objeto raster, banda 1 e shader acima. Em seguida aplicamos este ao objeto raster e chamamos a repintura do objeto

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

Se chamarmos o tipo novamente podemos ver a mudança.

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

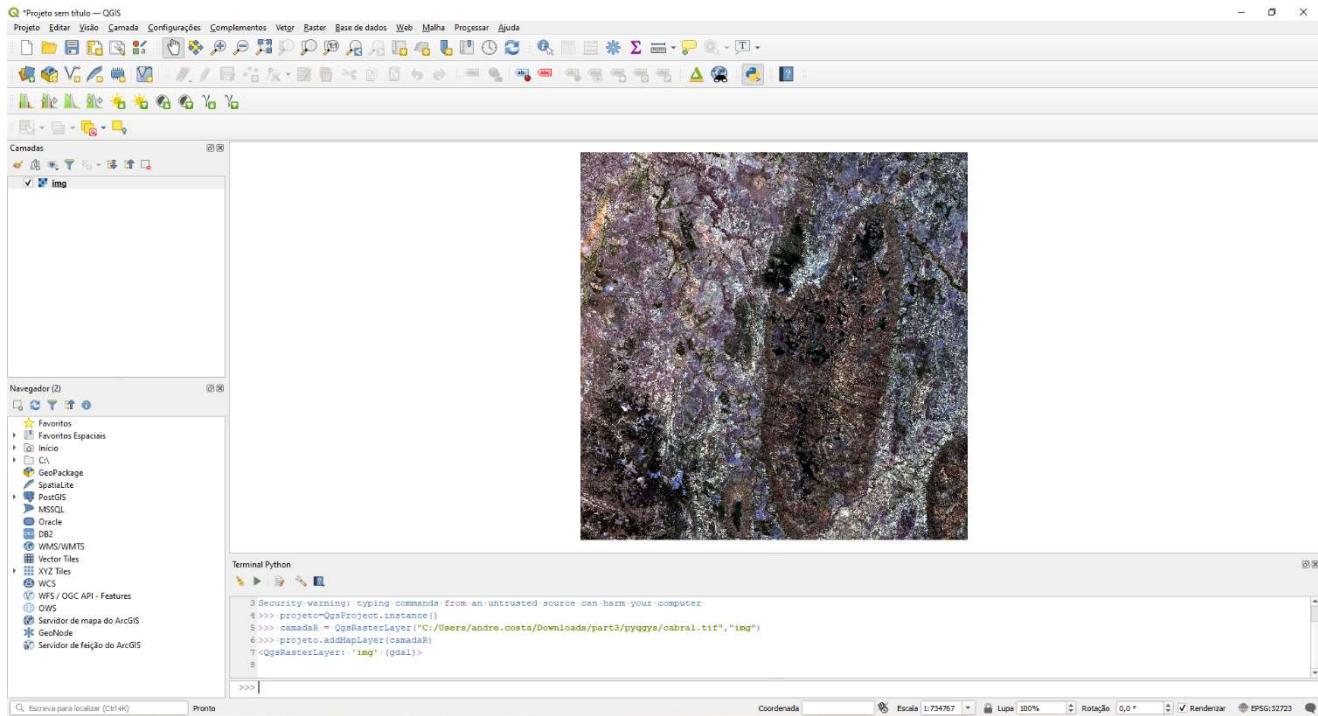
O resultado é mostrado num detalhe do raster (zoom in) na imagem abaixo.



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)
```



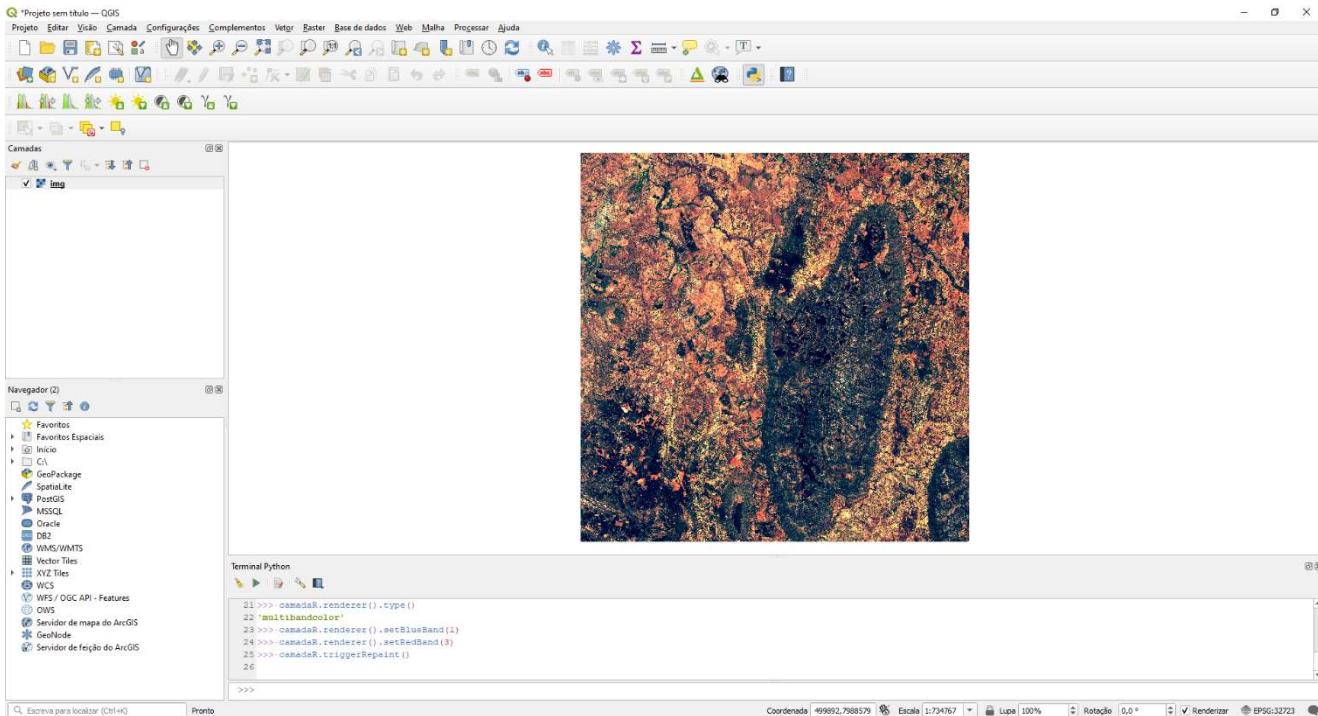
Podemos ver algumas das informações usando:

```
>>> 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 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)

```

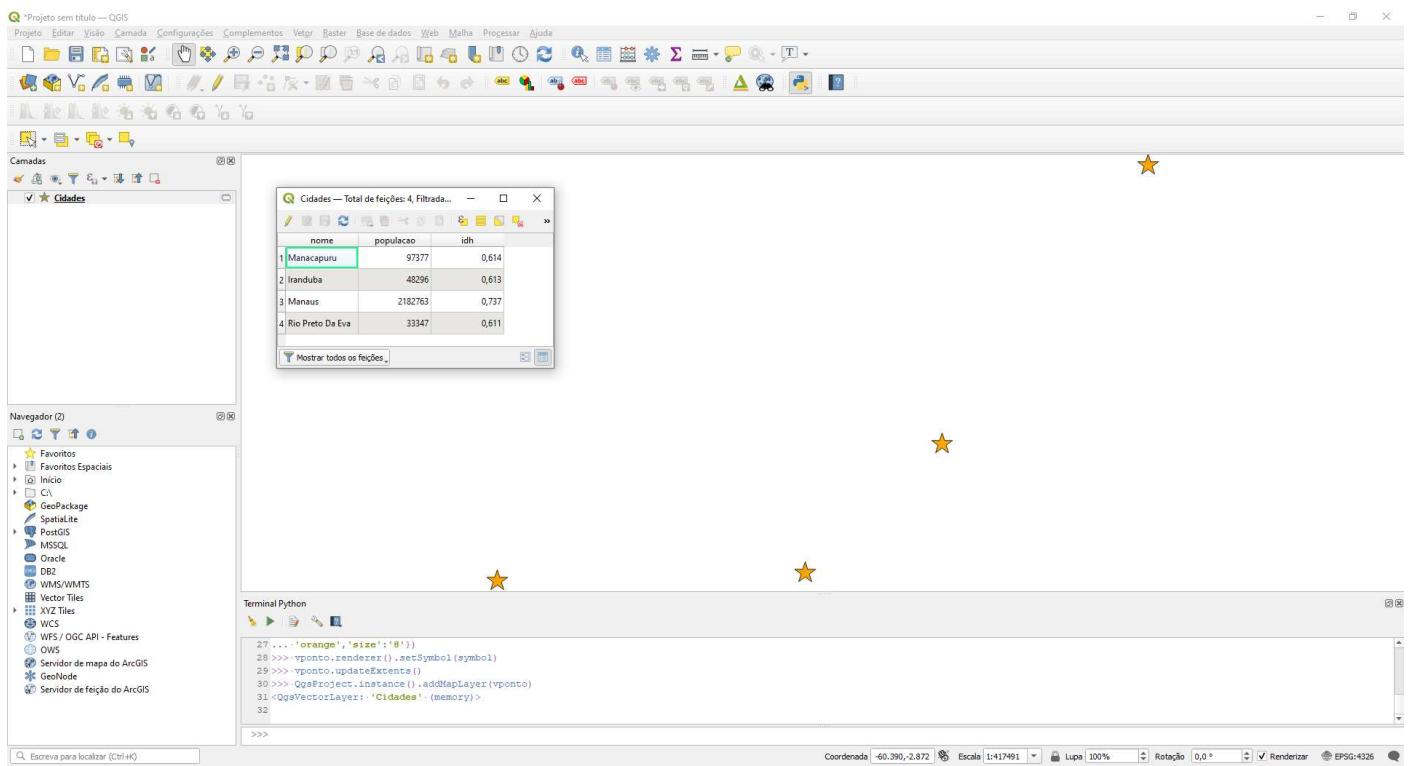
Para finalizar vamos configurar a aparência do símbolo mostrado no mapa como estrelas de cor laranjas e de tamanho 8. Atualizamos a extensão do mapa e adicionamos nosso objeto no mapa.

```

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

```

A aparência do mapa e de sua tabela de atributos 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'

```

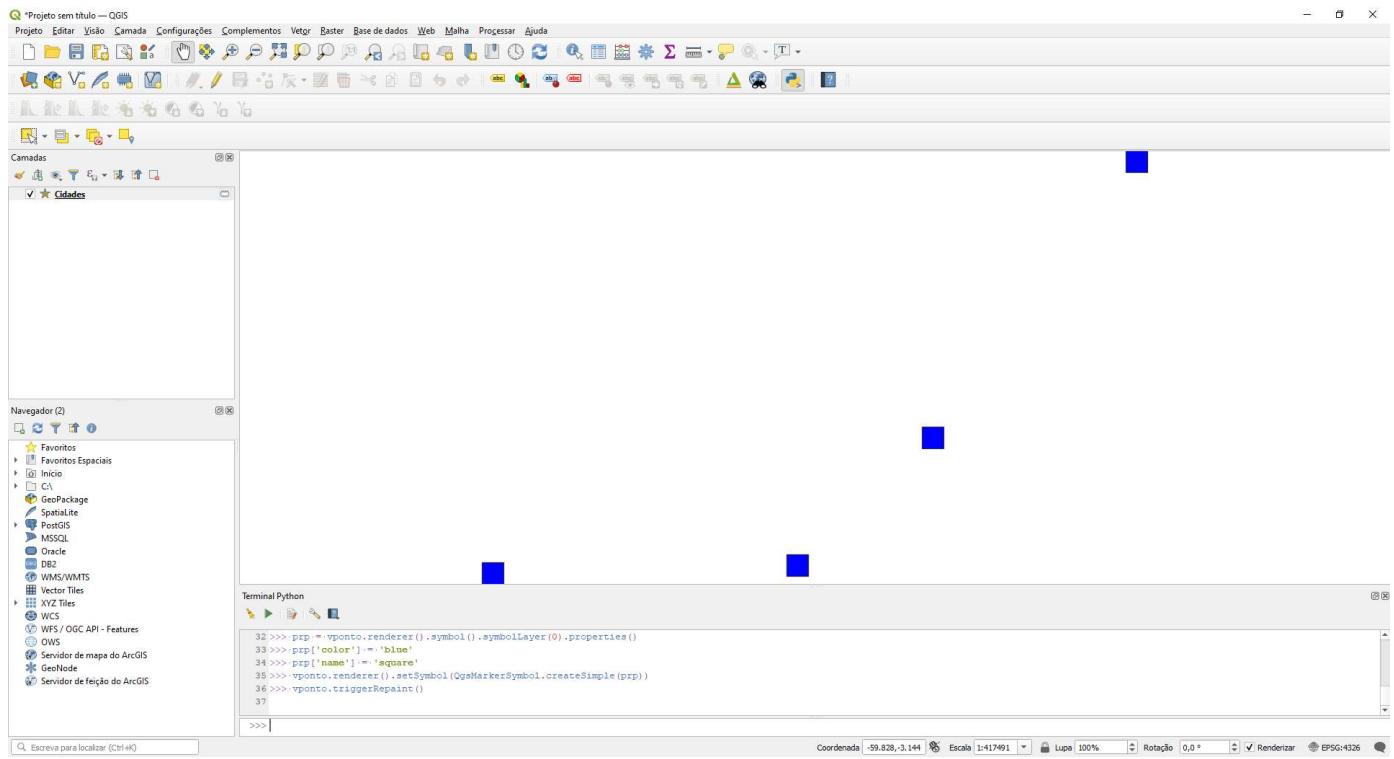
Veja a documentação para cada uma das opções que podem ser usadas com cada parâmetro listado acima. Vamos Modificar a aparência do símbolo que criamos acima.

```

>>> 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.

```

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

```

### 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).

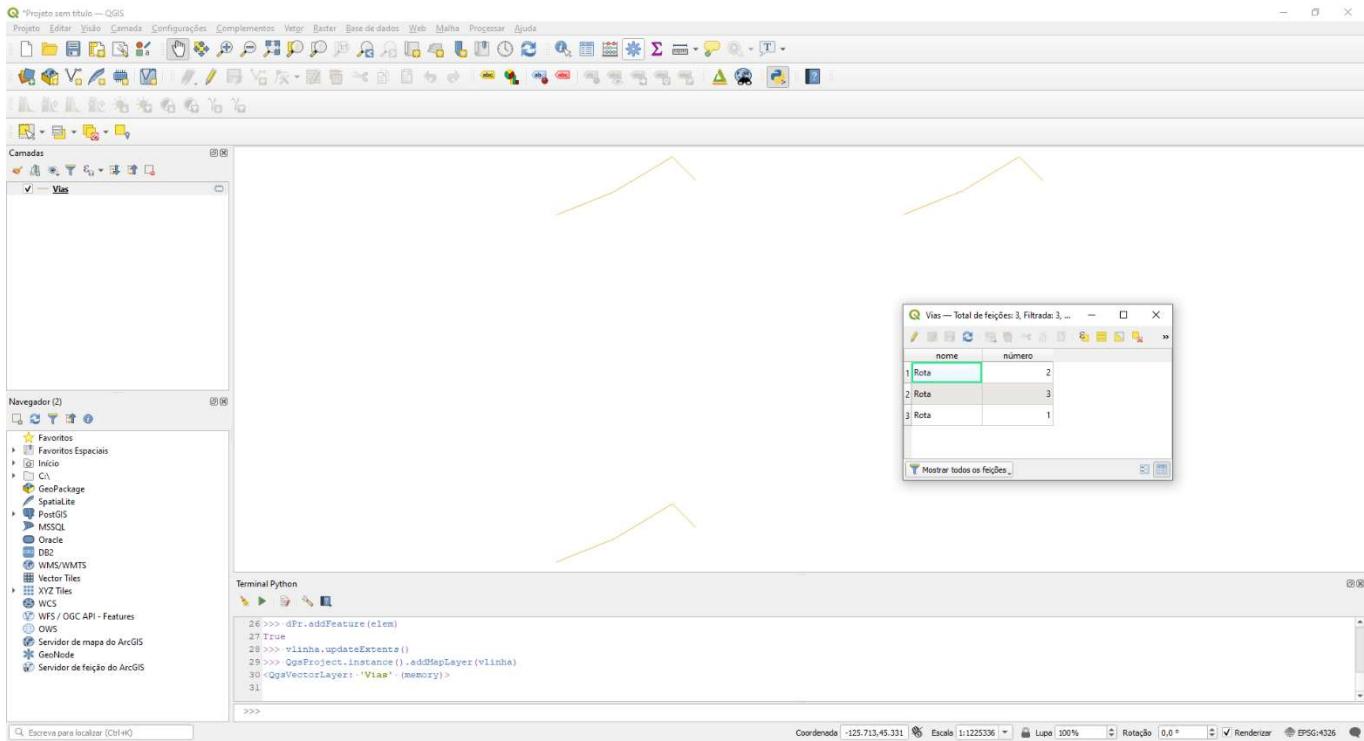
```
>>> vlinha = QgsVectorLayer("Linestring?crs=EPSG:4326", "Vias",
"memory")
>>> dPr = vlinha.dataProvider()
>>> dPr.addAttribute([QgsField("nome", QVariant.String),
QgsField("número", QVariant.Int)])
>>> vlinha.updateFields()
```

Adicionamos as linhas usando um objeto feature (elemento). Mas antes criamos a lista de pontos que farão parte de cada um dos elementos do tipo linha e 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)
```

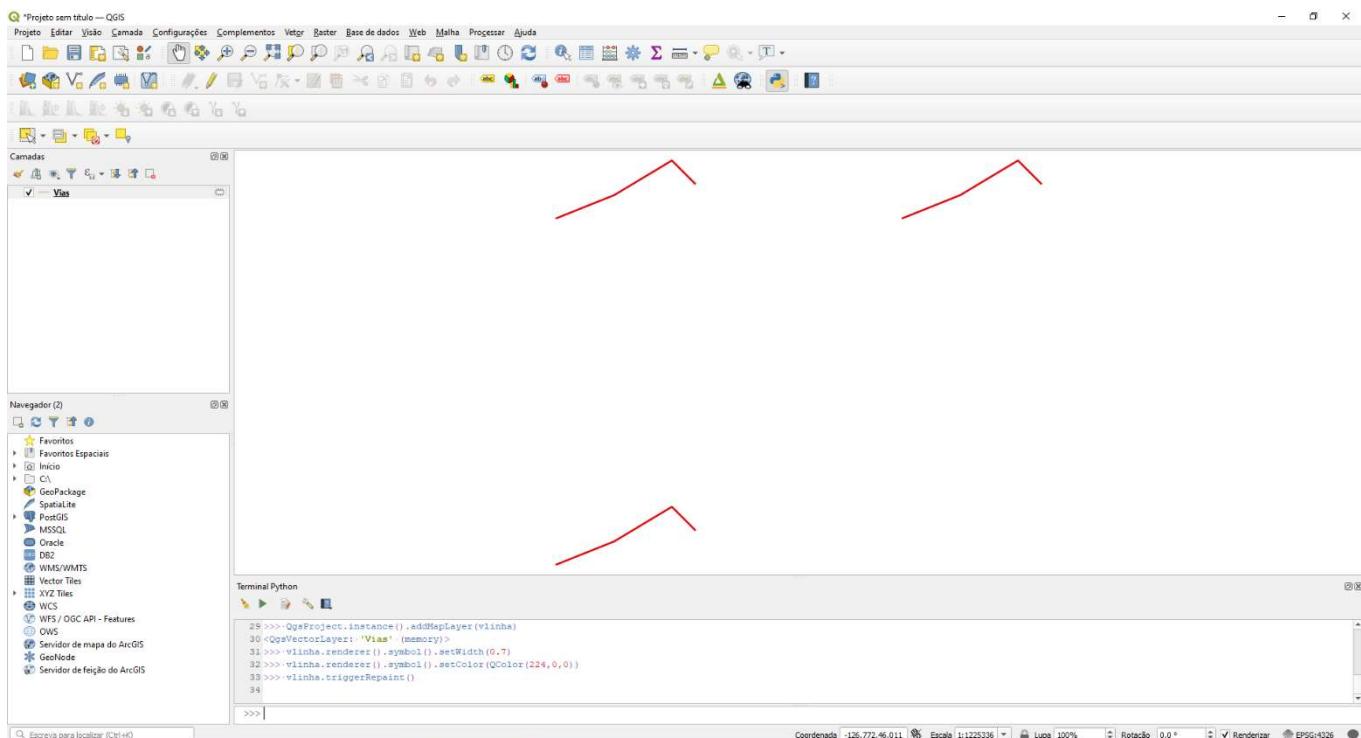
Atualizamos a extensão do objeto e o adicionamos ao mapa (canvas).

```
>>> 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 inserimos os atributos de cada elemento. Vamos inserir dois polígonos.

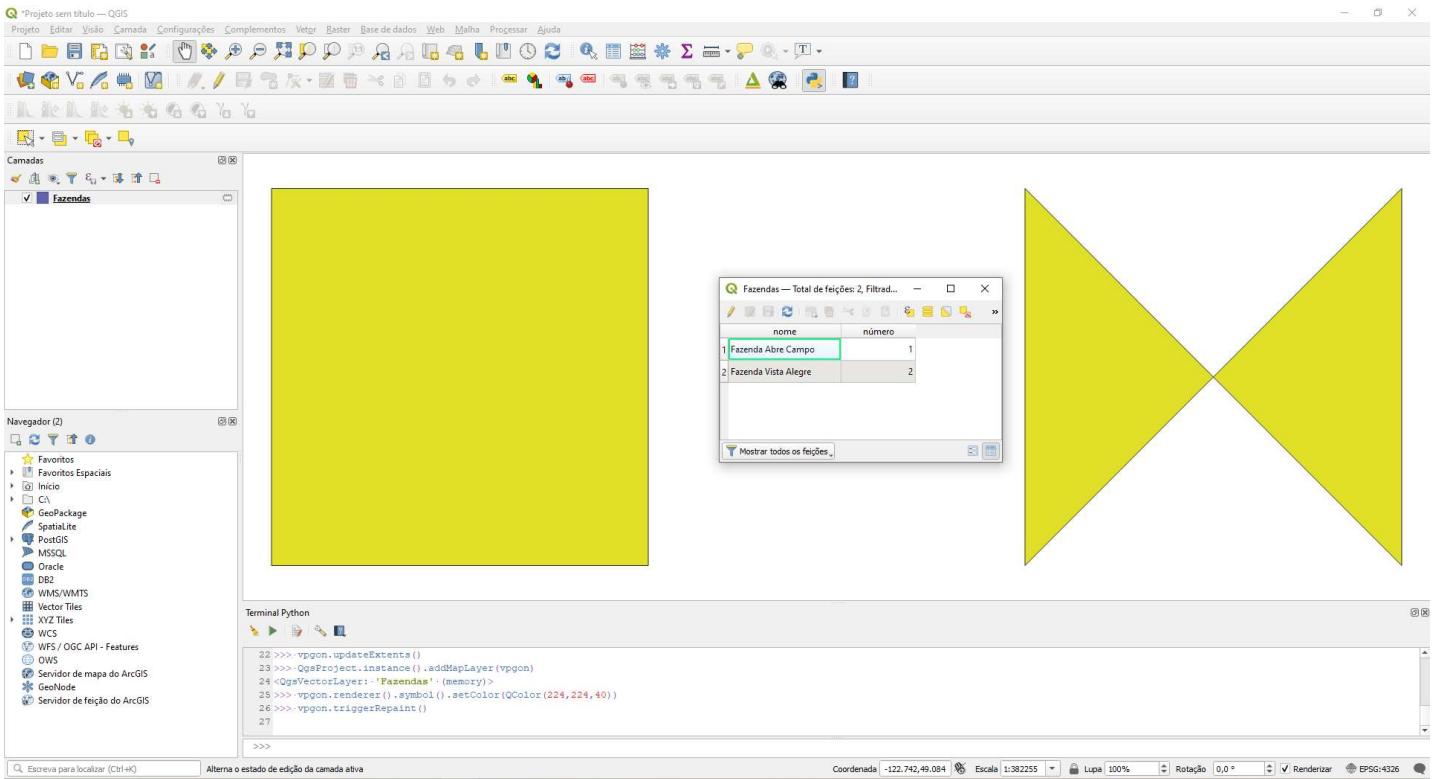
```
>>> elem = QgsFeature()
>>> pontos = [[QgsPointXY(-124, 48), QgsPointXY(-123, 48), QgsPointXY(-
123, 49), QgsPointXY(-124, 49)]]
>>> elem.setGeometry(QgsGeometry.fromPolygonXY(pontos))
>>> elem.setAttributes(["Fazenda Abre Campo ", 1])
>>> dPr.addFeature(elem)
>>> pontos = [[QgsPointXY(-122, 48), 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 criado e os parâmetros do grid a ser usado.

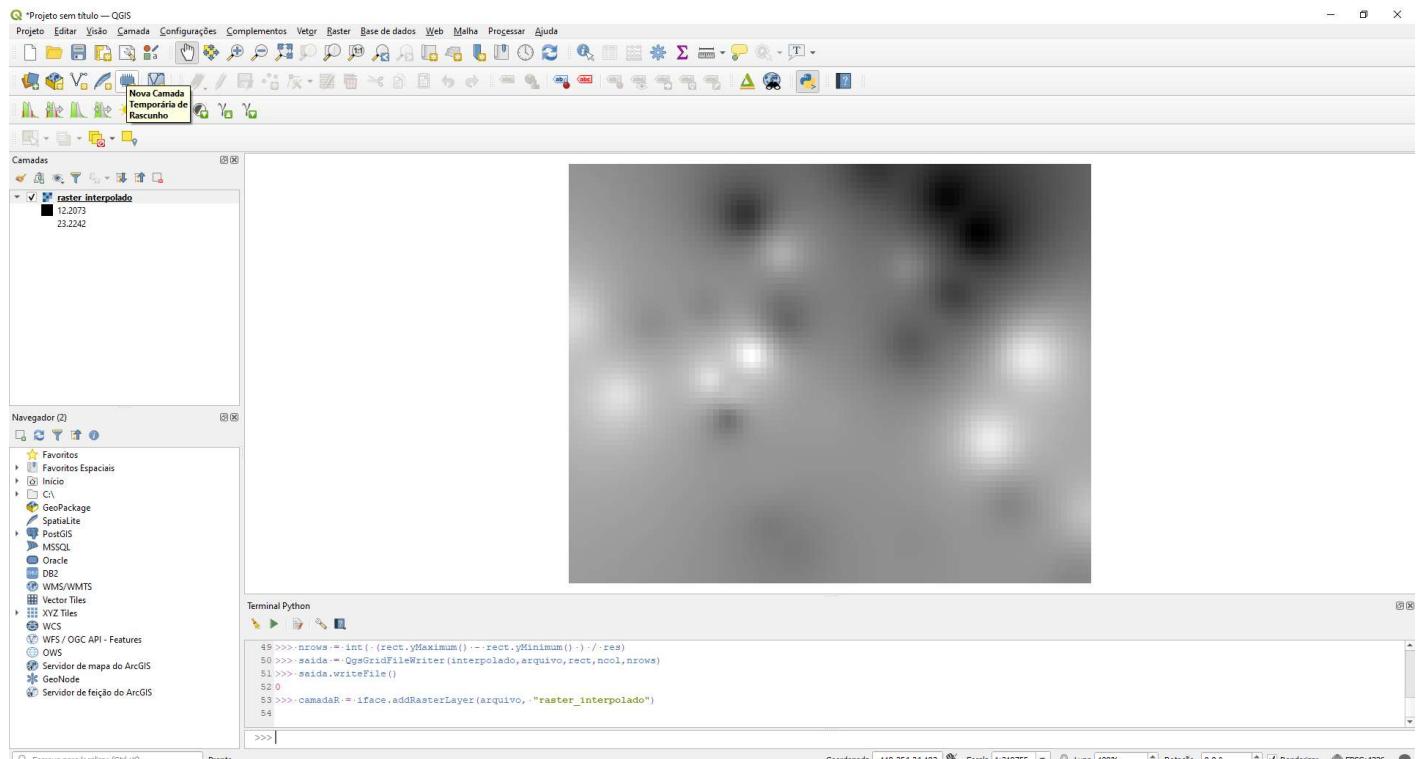
```

>>> 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")
```



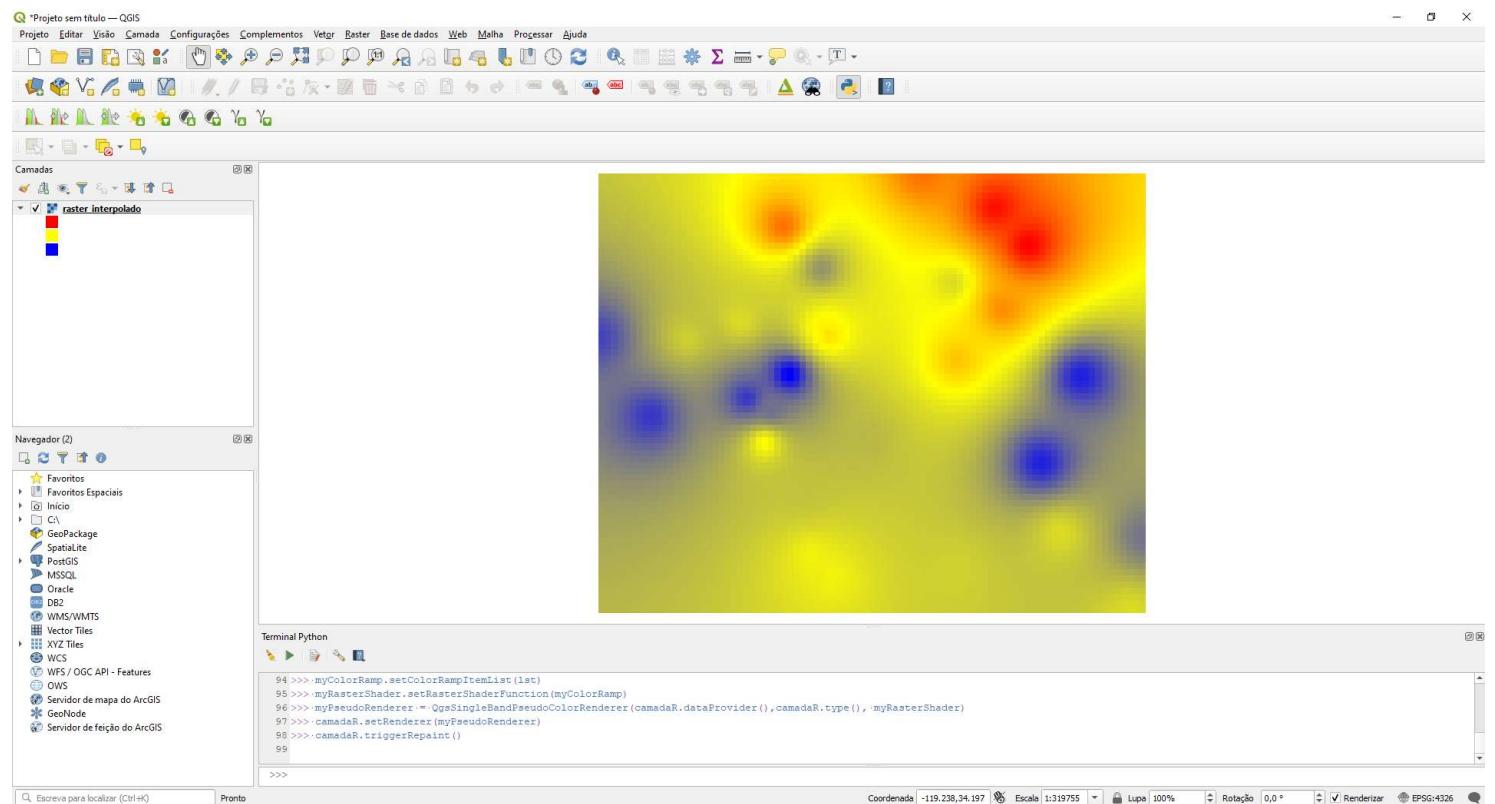
**Podemos alterar a aparência do raster que criamos usando o código seguinte.**

```

>>> 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],
QColor(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()
```

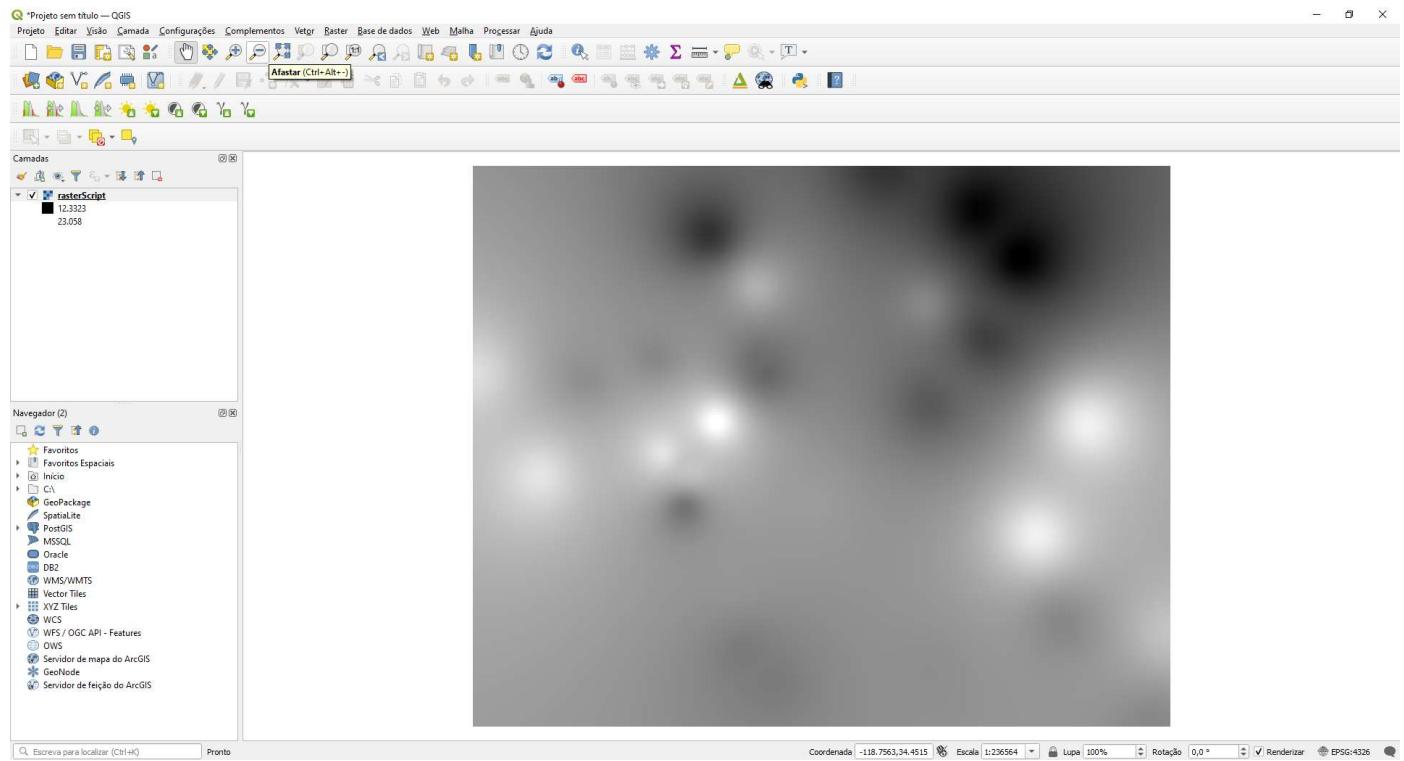
No ambiente linux/UNIX esse script funcionará sem problema usando **`python3 criaRaster.py`**, já para o ambiente Windows se faz necessário:

- Termos certeza que a variável PATH aponta para a pasta correta de instalação do Qgis, algo similar a e “C:/ProgramFiles/QGIS3.X/bin”

- Alterar a linha `QgsApplication.setPrefixPath("/usr", True)` para  
`QgsApplication.setPrefixPath("C:/ProgramFiles/QGIS3.X/bin/", True)`

- Executar usando `python-qgis-ltr criaRaster.py`

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



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