Capítulo 1

Google Earth Engine no Ensino de Introdução ao Sensoriamento Remoto e Geoprocessamento

Sérgio Souza Costa, Eduilson Lívio Neves da Costa Carneiro, Denilson da Silva Bezerra, Thomas Victor de Sousa Malheiros Rocha, Luis Fernando Cirqueira da Silva Correia

Abstract

This text discusses basic ideas about geoprocessing and remote sensing combined with practices using a cloud-based platform that offers data storage and computational processing. This approach helps avoid the need for non-specialized computer labs. To do this, we first introduce the Google Earth Engine platform and its main parts. Then, the various kinds of geographic data are presented within the platform's framework, allowing for the discussion of data selection and providing a few examples of processing afterward.

Resumo

Este texto aborda conceitos introdutórios sobre geoprocessamento e sensoriamento remoto integrado com práticas usando a plataforma em nuvem que disponibiliza armazenamento de dados e processamento computacional. Evitando assim a necessidade de laboratórios de informática não especializados. Para isso, inicialmente é apresenta a plataforma Google Earth Engine e seus principais componentes. Depois, os tipos de dados geográficos são apresentadas dentro do contexto da plataforma para que em seguida possa se apresentada a seleção e alguns exemplos de processamento.

1.1. Introdução

O tempo e o espaço são relevantes para a compreensão de diversos fenômenos geográficos naturais (ex: rios, formações rochosas, áreas de floresta) ou sociais como segregação, violência e epidemias. De acordo com [Longley 2005], os fenômenos geográficos são aqueles que possuem tempo e espaço bem definidos. O autor exemplifica essa ligação entre atributos, espaço e tempo com uma simples afirmação:

"A temperatura ao meio-dia local na altitude 34 *graus e* 35 *minutos norte, longitude,* 120 *graus* 0 *minutos oeste, era* 19 *graus Celsius".*

O estudo de fenômenos geográficos fazem parte da ementa de disciplinas de diversos cursos, principalmente na área de Geociências. Nos primeiros semestres é comum a oferta de uma ou duas disciplinas de introdução ao geoprocessamento e/ou sensoriamento remoto. Estas disciplinas requerem laboratórios de informática equipados com soluções de software proprietário como ArcGIS[1] ou abertos como QGIS[2] e Spring[3].

Os custos na montagem destes laboratórios podem ser reduzidos com soluções abertas e gratuitas. Contudo, ainda seria necessário a aquisição de computadores com alto poder computacional, espaço de armazenamento e uma boa conectividade para a realização de diversas práticas, principalmente para aquelas que envolvam processamento de imagens de satélite. Que além de poder computacional requerem espaço em disco e tempo para poder baixar diversas imagens. Acredita-se que muitos dos laboratórios de informática nas universidades brasileiras não possuem estas características. Então, este trabalho propõe a utilização de uma solução que possa aproveitar melhor os laboratórios existentes equipados com computadores de baixo poder computacional e sem a necessidade de instalação de software especializado.

A proposta se fundamenta na utilização de uma plataforma computacional baseada na nuvem, mais especificamente, o Google Earth Engine (GEE) [Gorelick et al. 2017]. Esta plataforma concentra um grande volume de algoritmos e de dados provenientes de diferentes fontes como modelo digital de elevação, séries históricas de imagens de diversos programas espaciais. Essa abordagem evita a necessidade de baixar um grande volume de dados, além de migrar todo o processamento para uma arquitetura computacional na nuvem. Deste modo, este minicurso pretende demonstrar como essa plataforma pode ser usada por docentes de diversos cursos e formação em suas disciplinas introdutórias de geoprocessamento e sensoriamento remoto.

1.2. Conhecendo o Google Earth Engine (GEE)

A plataforma Google Earth Engine (GEE) representa uma infraestrutura de processamento e análise de imagens geoespaciais, concebida e mantida pela corporação Google, voltada para a realização de investigações científicas e análises geoespaciais avançadas. Ela combina uma vasta coleção de imagens de satélite e dados geoespaciais com recursos de processamento em nuvem e ferramentas de análise para permitir que os usuários estudem e visualizem informações em escala global [Gorelick et al. 2017]. Dentre as principais características destacam-se: extenso catálogo de imagens de satélite, algoritmos de processamento, processamento em nuvem e visualização interativa.

O GEE disponibiliza um extenso catálogo de imagens de satélite provenientes de diversas fontes, incluindo as missões Landsat, Sentinel e MODIS. Tal abrangência de fontes de dados permite aos pesquisadores e usuários acessarem informações tanto atuais quanto históricas, permitindo, assim, o monitoramento de alterações na superfície terres-

[1] https://www.arcgis.com/
[2] https://qgis.org/
[3] https://www.dpi.inpe.br/spring/

tre ao longo do tempo. O GEE disponibiliza uma variada gama de algoritmos prontos e acessíveis para o processamento de dados matriciais e vetoriais, incluindo ferramentas de classificação de imagens. Tais algoritmos são executados na infraestrutura de servidores da Google, aliviando os usuários das preocupações inerentes à aquisição e à gestão de vastos conjuntos de dados. Além da exigência de capacidade computacional necessária para o processamento dessas informações. A plataforma oferece também recursos de visualização interativa que facultam aos utilizadores a criação de mapas interativos, bem como a personalização de apresentações e painéis com vistas à exploração e ao compartilhamento dos resultados obtidos.

Além destes recursos, a plataforma apresenta uma diversidade de aplicações, abrangendo áreas como o monitoramento ambiental, a gestão de recursos naturais, a agricultura de precisão e a detecção de desastres naturais, dentre outras. Pesquisadores, cientistas, desenvolvedores e entidades diversas têm aproveitado a plataforma para a realização de análises geoespaciais avançadas. A título de exemplo, no contexto brasileiro, destaca-se o projeto MapBiomas [MapBiomas 2021], que ilustra a utilização bem-sucedida do GEE. Estas aplicações e algoritmos estão integradas mediante bibliotecas disponíveis para as linguagens JavaScript e Python. Nesse texto, iremos usar as bibliotecas em JavaScript por estarem disponíveis para a utilização através do portal `https://code.earthengine.google.com/`, como mostra a Figura 1.1.

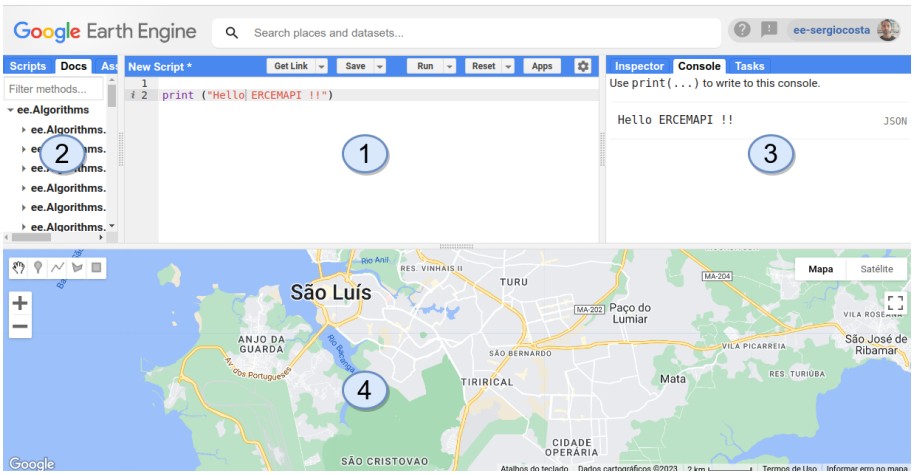


Figura 1.1. Editor de Código do GEE

A Figura 1.1 apresenta o editor destacando 4 principais divisões:

1. **Área de Código**: nesse espaço, os *scripts* são desenvolvidos para carregar, processar e visualizar imagens e dados geoespaciais. O editor comumente oferece realce de sintaxe para facilitar a escrita e a correção de erros no código.

2. **Abas à Esquerda**: nesse espaço estão disponíveis três abas que exibem os Scripts organizados em arquivos e pastas, Docs com a documentação dos principais comandos e funções e os Assets com acesso aos dados vetoriais e matriciais do usuário.

3. **Abas à Direita**: nesse espaço são encontradas abas ou guias que oferecem acesso a funcionalidades adicionais. Incluindo guias para os resultados da execução de um

script, gráficos gerados, registros de saída e progresso de tarefas de importação e exportação.

4. **Área do Mapa**: nesse espaço é exibido um mapa interativos, onde é possível definir parâmetros de visualização, seleção de camada e desenho de dados vetoriais.

O Google Earth Engine foi projetado para poder ser usado por profissionais de diversas áreas sem experiência com programação. Contudo, a melhor utilização desta plataforma pode ser alcançado com a compreensão de alguns conceitos que serão descritos na Seção 1.2.1.

1.2.1. JavaScript no Google Earth Engine

A linguagem JavaScript foi criada por Brendan Eich em 1995 na Netscape com objetivo inicial de ser executada nos navegadores web tornando as páginas mais interativas [Wirfs-Brock and Eich 2020]. Desde então, ela evoluiu e atualmente não está limitada apenas aos navegadores. Antes de ser escolhida para acessar as funcionalidades da plataforma do Google Earth Engine, ela já tinha sido escolhida como a linguagem de script para integrar as aplicações de escritório como Google Docs e Google Spreadsheet [Ferreira 2014]. É importante ressaltar que um *script* consiste em uma sequência de comandos que um computador segue para realizar determinado processo, análogo a uma receita culinária. Portanto, é imprescindível ter conhecimento acerca das instruções ou comandos individuais e compreender a maneira adequada de combiná-los.

Existem algumas instruções que apenas causam um efeito, como a impressão de dados no console (aba à direita da Figura 1.1) ou executa uma operação de zoom na área do mapa, como no código a seguir:

```
1 print ("Hello ERCEMAPI !!")
2 Map.setCenter(-44.2162345711303,-2.548309158409205,12)
```

Enquanto outras instruções geram novos valores a partir de uma busca em uma base de dados ou a partir do processamento de valores criados previamente. Por exemplo, a instrução `ee.Image` busca um objeto do tipo imagem no catálogo a partir de um identificador. Nestes casos, é comum associar esses objetos (ou valores) retornados a um nome, ou seja, armazena-se o resultado das instruções em variáveis. Por exemplo, pode-se associar uma dada imagem ao nome "saoluis" como no exemplo a seguir.

```
1 var saoluis = ee.Image ("LANDSAT/LC08/C01/T1_TOA/LC08_220062_20190504")
```

O uso de variáveis é necessário para poder referenciar esses nomes durante cada um dos três principais passos exemplificados na Figura 1.2. Na **Seleção/Entrada** é criado um objeto do tipo imagem a partir de um identificador e guardado na variável `javascript`. Esse objeto passa por um **Processamento** gerando um novo objeto que será então guardado na variável `javascript`. Essa variável é então adicionado ao mapa para poder ser enviada para a **Apresentação/Saída**.

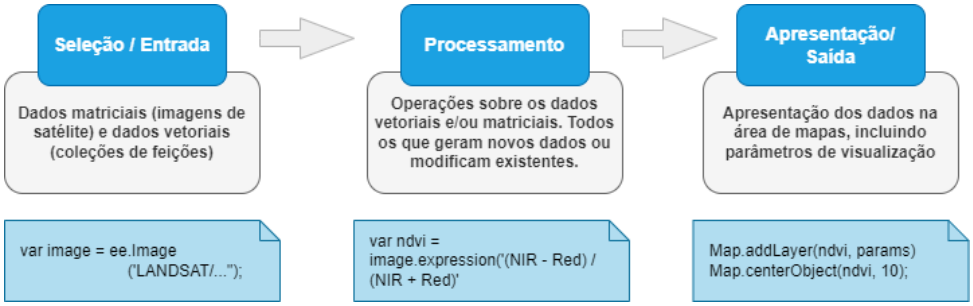


Figura 1.2. Os três principais passos em um *script*

Em algumas situações, pode ser interessante agrupar mais de um valor a mesma variável (ou nome). Como, por exemplo, as informações (metadados) de uma imagem de satélite:

```
var metadados = {
  dataAquisicao: '2019-09-15',  resolucaoEspacial: 30,
  sistemaCoordenadas: 'EPSG:4326'
};
```

Com estas informações agrupadas em um mesmo nome, é possível acessá-las como a seguir:

```
print('Data de Aquisição:', metadados.dataAquisicao);
print('Resolução Espacial:', metadados.resolucaoEspacial);
```

Neste exemplo, a variável `metadados` é um dicionário que armazena informações sobre a imagem, facilitando acessar essas informações quando necessárias. Dicionários são fundamentais e serão usados na definição dos atributos das feições geográficas e nos parâmetros de algumas instruções, como na visualização.

O processamento ilustrado na Figura 1.2 é executado mediante uma única instrução. Contudo, em numerosas situações, o processamento requererá várias instruções, as quais podem ser organizadas e encapsuladas por meio de funções que dado uma entrada, ela será então processada e depois retornada, como ilustrado na Figura 1.3.

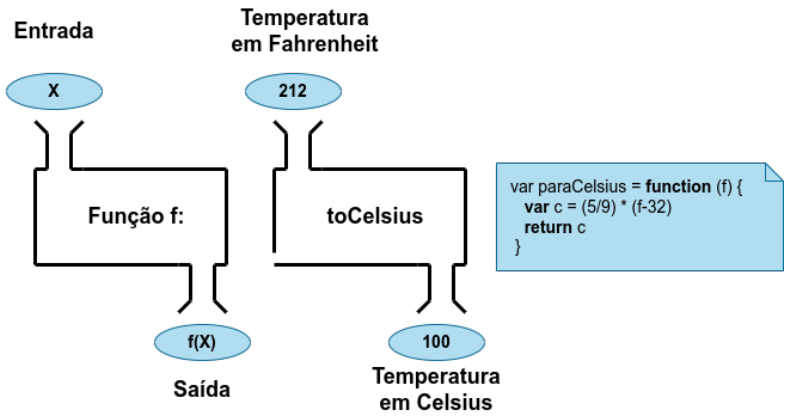


Figura 1.3. Conceito e exemplo de funções

Na Figura 1.3, o lado esquerdo representa o conceito fundamental de uma função, que envolve três elementos essenciais: entrada, processamento e saída. O lado direito da Figura 1.3 ilustra um exemplo concreto de uma função projetada para converter uma temperatura dada em graus Fahrenheit em sua equivalente em graus Celsius, realizando o cálculo e fornecendo o resultado como saída.

1.3. Tipos de dados geográficos

Os tipos de dados geográficos desempenham um papel fundamental na análise espacial e na tomada de decisões em uma variedade de campos, desde a geografia até a ciência ambiental. Como mencionado em [Smith et al. 2007], esses dados podem ser categorizados em dois tipos principais: dados vetoriais e dados matriciais (referido em algumas referências como *raster*). Os dados vetoriais representam objetos geográficos como pontos, linhas e polígonos, enquanto os dados matriciais dividem o espaço em células regulares com valores associados.

1.3.1. Dados vetoriais

Nos dados vetoriais, a localização e a forma de cada objeto são representados por um ou mais pares de coordenadas espaciais (geometrias), podendo incluir atributos não-espaciais que os descrevem e identificam univocamente [DAVIS and Fonseca 2001]. Esses pares de coordenadas são usadas para representar geometrias simples como ponto, linha e polígono (área), Figura 1.4.

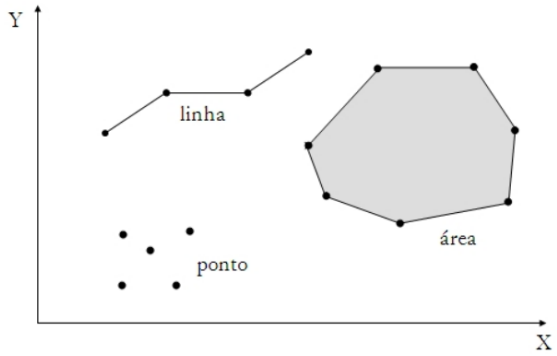


Figura 1.4. Tipos geométricos primitivos. Fonte: [Câmara 2005]

Estas três primitivas podem ser representadas usando diferentes formatos, mas nesse texto será usado formato WKT (*Well-Known Text*) por ser um formato mais legível ajudando a melhor compreender sua estrutura interna. Esse formato inclue informações sobre o tipo do objeto e as coordenadas (longitude e latitude) que o formam, ou seja, é o nome da representação seguido de pares de coordenada. Por exemplo, as três primitivas seriam representadas como:

- POINT(-44.29777178028425 -2.5313470294034044)
- LINESTRING(-44.300075363904284 -2.543551794770009, ... ,-44.338527512341784 -2.564387934650711)

- POLYGON ((-44.29622682789167 -2.5235440348039972, ... ,-44.29622682789167 -2.5235440348039972))

O GEE suporta essas primitivas de modo similar, através dos tipos de dados `ee.Geometry.Point`, `ee.Geometry.LineString` e `ee.Geometry.Polygon` com os pares de coordenadas separadas por vírgulas e entre colchetes. Essas geometrias podem ser criadas especificando os pares de coordenadas, através do editor de geometrias na área de mapa ou importadas a partir de fontes externas. O código a seguir mostra como um ponto poderia ser criado especificando seus pares de coordenadas:

```
1 // WKT - POINT(-44.29777178028425 -2.5313470294034044)
2 var geo = ee.Geometry.Point([-44.29777178028425, -2.5313470294034044]);
```

O código anterior já poderia ser executado na plataforma, contudo para visualizar os dados é necessário adicioná-lo ao mapa e aplicar uma ampliação até ele[1]:

```
1 // passo 1: criação do objeto ponto
2 var geo = ee.Geometry.Point([-44.29777178028425, -2.5313470294034044]);
3 // passo 2: este exemplo não teve nenhum processamento
4 // passo 3: apresentação
5 Map.addLayer(geo , {},'Em algum lugar em São Luís');
6 Map.centerObject(geo, 15);
```

As entidades geográficas codificadas usando o modelo de dados vetoriais são denominadas de feições geográficas (*features*). As feições são objetos vetoriais dos tipos ponto, linha ou polígono [Longley 2005]. Além da forma geométrica, elas são associadas a outros atributos. Por exemplo, uma feição poligonal que representa um município pode ter dados de atributos que incluem o nome do município, população e índice de desenvolvimento humano.

No Google Earth Engine, uma feição é um objeto que consiste em uma geometria e um conjunto de atributos associados. Estes atributos são geralmente representadas como um dicionário[2] de chave-valor, onde cada chave é um nome que descreve aquele atributo e o valor pode ser de qualquer tipo de dado. Por exemplo, o Código 1.1 cria uma geometria com um polígono representando os limites da Universidade Federal do Maranhão e dois atributos. Observe que na definição dos atributos as chaves foram **nome** e **sigla** tendo respectivamente os valores "Universidade Federal do Maranhão" e "UFMA".

```
1 var geo = ee.Geometry.Polygon(
2         [[[-44.305962613096874, -2.5505973845937544],
3           [-44.31527524279658, -2.5613584194753107],
4           [-44.308923771849315, -2.564230871681558],
5           [-44.30261521624629, -2.5549275329820973]]]);
6 var attrs = {nome: "Universidade Federal do Maranhão", sigla:"UFMA"}
7 var feature = ee.Feature(geo, attrs )
8 Map.addLayer(feature)
9 Map.centerObject(feature, 16)
```

Código 1.1. Exemplo de uma simples feição geográfica (*feature*)

[1]Observe que este seria o passo 3 (apresentação/saída) como descrito na Seção 1.2.1

[2]Observe aqui uma aplicação do dicionário citado na Seção 1.2.1

Usuários de sistemas de informação geográfica estão habituados a trabalhar com coleções de feições. Por exemplo, o mapa do estado do Maranhão é uma coleção de feições onde cada município é uma feição. No GEE, esse conceito é reprsentado pelo tipo de dados `ee.FeatureCollection` que é basicamente um conjunto de `ee.Feature`. Essa coleção ser criadas a partir de uma lista de feições (Código 1.2), podem ser carregados diretamente do catálogo plataforma ou importados de outras fontes.

```
1  var s1 = ee.Geometry.Point([-44.28622329382671,-2.5107472461135463]);
2  var s2 = ee.Geometry.Point([-44.25510924089217,-2.527503498708386]);
3  var s3 = ee.Geometry.Point([-44.22563878039214,-2.5338115592006343]);
4
5  var shoppingCollection = ee.FeatureCollection([
6    ee.Feature(s1, { nome: 'Shopping São Luís' }),
7    ee.Feature(s2, { nome: 'Shopping da Ilha' }),
8    ee.Feature(s3, { nome: 'Shopping Rio Anil' })
9  ]);
10
11 Map.addLayer(shoppingCollection , {},'Maiores shoppings');
12 Map.centerObject(shoppingCollection, 10);
```

Código 1.2. Exemplo simples de uma coleção de feições geográficas (*feature*)

O Código 1.2 apresentou um exemplo simples onde foram definidos três feições representando os principais shoppings do município de São Luis do Maranhão. Neste caso, a coleção foi criada explicitamente por uma lista com as 3 feições. Contudo, o mais comum é que estas coleções venham de outras fontes de dados como será visto na Seção 1.4.

1.3.2. Dados matriciais

Em uma representação matricial o espaço é divido em uma malha retangular de células (geralmente quadradas). Nessa representação, o espaço é concebido como uma matriz $P(m,n)$ com m colunas e n linhas, em que cada célula possui um número de linha, um número de coluna e um valor correspondente ao atributo em análise [Câmara 2005]. Um exemplo comum de dados matriciais são as imagens geradas a partir dos sensores a bordo de satélites ou aerotransportado. A resolução espacial destes dados depende da relação entre o tamanho da célula e a área que ela representa no terreno, Figura 1.5.

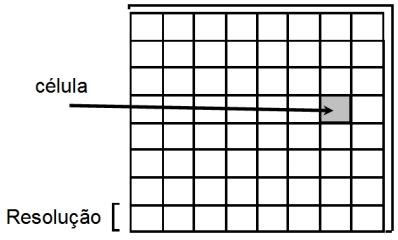


Figura 1.5. Dados matriciais

Geralmente, a resolução espacial é expressa em unidades de comprimento, como metros ou centímetros. Quanto menor for esse valor, maior será a resolução espacial, o que significa que a imagem será capaz de capturar detalhes menores. Por exemplo, na Figura 1.5 apresenta uma dada cena em diferentes resoluções espaciais. Além da resolução

espacial, é importante conhecer outros tipos de resolução, como a temporal, espectral e radiométrica. A resolução temporal se refere ao intervalo que uma dada imagem e capturada, por exemplo, tem satélites que passam regularmente sobre um dado ponto a cada 15 dias[1]. Esse intervalo define a resolução temporal. A resolução espectral é a capacidade de um sensor remoto de dividir o espectro eletromagnético em várias bandas espectrais ou canais de detecção. Cada banda representa uma faixa específica de comprimentos de onda da luz visível, infravermelha, entre outras. Essa divisão do espectro permite que o sensor detecte e registre informações em diferentes partes do espectro eletromagnético.

Uma imagem de sensoriamento remoto pode ser composta por várias bandas espectrais, que são sensíveis as diferentes faixas do espectro eletromagnéticos como vermelho, verde e infravermelho próximo. Cada banda captura informações em uma faixa específica do espectro. A Figura 1.6 representa a faixa do espectro eletromagnético e as bandas do Landsat 9 em cada faixa.

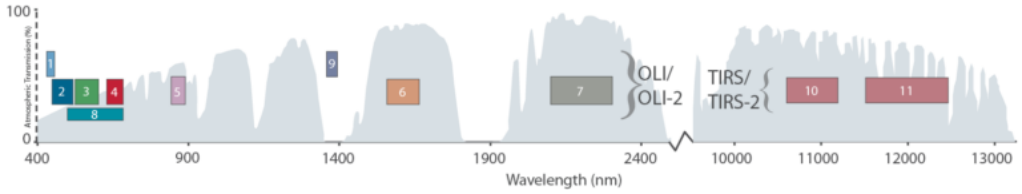


Figura 1.6. O espectro eletromagnético.

A resolução radiométrica refere-se à capacidade de um sensor em registrar variações na intensidade da radiação eletromagnética (geralmente na forma de energia refletida ou emitida) em suas várias bandas espectrais. Em termos mais simples, é a capacidade do sensor de distinguir entre tons de cinza em uma imagem. Por exemplo, em uma imagem com resolução radiométrica de 8 bits, existem 256 tons de cinza diferentes que podem ser representados, variando de preto (valor mínimo) ao branco (valor máximo).

O Google Earth Engine oferece uma variedade de recursos para trabalhar com dados de sensoriamento remoto através de dois tipos de dados: `ee.Image` e `ImageCollection`. O tipo de dados `ee.Image` representa uma única imagem de satélite que pode ser composta por várias bandas espectrais, cada uma representando informações em uma faixa específica do espectro eletromagnético (por exemplo, vermelho, verde, infravermelho próximo, etc.), como ilustrado na Figura 1.6. Cada *pixel* em uma imagem possui valores de intensidade para cada banda espectral, e os dados geoespaciais são associados a coordenadas de localização e sistema de projeção. Similar a `ee.FeatureCollection`, pode-se carregar uma imagem a partir do seu identificador como no Código 1.3.

```
1 var saoluis = ee.Image (
2     "LANDSAT/LC08/C01/T1_TOA/LC08_220062_20190504" )
3 // os parametros de visualização serão explicados posteriormente
4 var vizparams = {bands: ["B4", "B3", "B2"], min: 0, max: 0.5}
5 Map.addLayer(saoluis, vizparams)
6 Map.centerObject(saoluis, 8)
```

Código 1.3. Carregando uma imagem a partir do seu identificador

[1]Existem também os satélites geoestacionários que estão sempre sobre um ponto fixo

No Código 1.3 a variável `saoluis` representa uma única imagem Landsat 8 capturada em 04 de maio de 2019, na região com valor 200 para o *path* e 062 para *row*[1]. Essa imagem contém várias bandas espectrais, como vermelho, verde, infravermelho próximo, entre outras, que podem ser usadas para análises espectrais. Com base nas diferentes respostas espectrais, é possível discernir os diferentes alvos sobre a superfície terrestre, estimar temperaturas entre outras aplicações.

Uma `ee.ImageCollection` é uma estrutura de dados que contém várias imagens organizadas por data de aquisição, localização ou outros critérios. Ela permite representar conjuntos de dados geoespaciais em diferentes momentos, sendo crucial para análises temporais no Google Earth Engine.

```
1 var saoLuis = ee.Geometry.Point(-122.081, 37.419)
2
3 var colecao = ee.ImageCollection('LANDSAT/LC09/C02/T1')
4 .filterDate('2022-01-01','2022-10-30')
5 .filterBounds(saoLuis)
6
7 print("Quantidade de Imagens: ", colecao.size().getInfo())
```

Código 1.4. Exemplo simples de uma feição geográfica (*feature*)r

Com `ee.ImageCollection` é possível fazer análises temporais, como detecção de mudanças e identificação de tendências sazonais. Reduções de ruídos, alcançado através da obtenção de uma média de múltiplas imagens, o que resulta em resultados mais suaves e confiáveis, minimizando assim os efeitos indesejados de variações aleatórias nos dados. Além de operações em lote, dado que a plataforma permite a aplicação de operações em toda a coleção de imagens de maneira conjunta. Isso simplifica consideravelmente a condução das análises, resultando em maior eficiência e produtividade.

1.4. Seleção e visualização de dados geográficos

Nas seções anteriores foram apresentados alguns conceitos básicos da plataforma e sobres os tipos de dados geográficos. Nesta seção será visto melhor como selecionar os dados de interesse.

1.4.1. Seleção e visualização de dados vetoriais

Na Seção 1.3.1 foram apresentados os tipos de dados vetoriais usados na representação computacional dos objetos discretos dentro dos sistemas de informação e banco de dados geográficos. Nessa Seção será apresentado como carregar estes dados diretamente da plataforma do Google Earth Engine ou de outras fontes.

1.4.1.1. Acessando os dados vetoriais

O catálogo do Google Earth Engine é predominantemente composto por dados matriciais. No entanto, também inclui algumas coleções de dados vetoriais, como informações sobre limites de países e outros conjuntos de dados fornecidos principalmente por agências dos

[1] As coordenadas *path* e *row* permitem especificar uma imagem Landsat única em um determinado local na Terra.

Estados Unidos. Para encontrar estes dados, pode-se buscar o seu identificador (ID) no catálogo como na Figura 1.7.

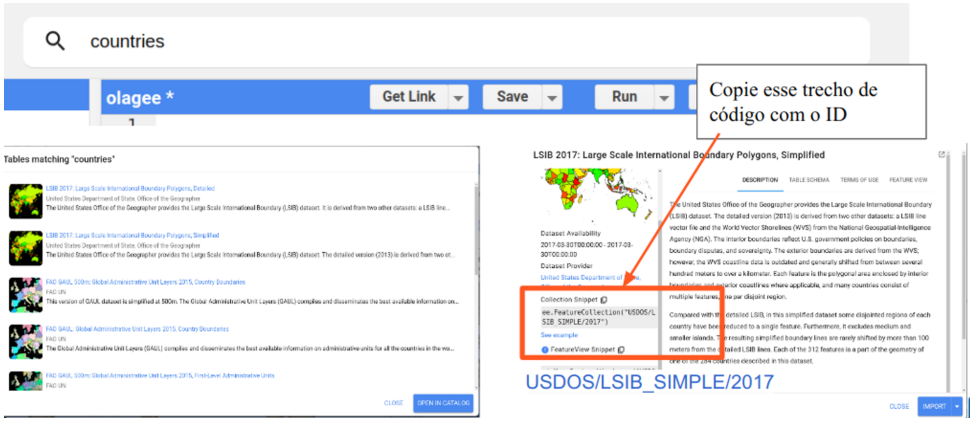


Figura 1.7. Buscando dados vetoriais no catalogo

Com esse identificador, é possível carregar essa coleção de feições como no código a seguir:

```
1 var paises = ee.FeatureCollection("USDOS/LSIB_SIMPLE/2017")
```

Conforme discutido anteriormente, o catálogo atualmente apresenta um número limitado de coleções de dados vetoriais. No entanto, é possível importar informações vetoriais de outras fontes, desde que estejam no formato Shapefile. No portal do Instituto Brasileiro de Geografia e Estatística (IBGE) é possível baixar malhar territoriais de todo país em formato Shapefile que poderão ser transferidas para a plataforma do Google Earth Engine, seguindo os procedimentos detalhados na Figura 1.8.

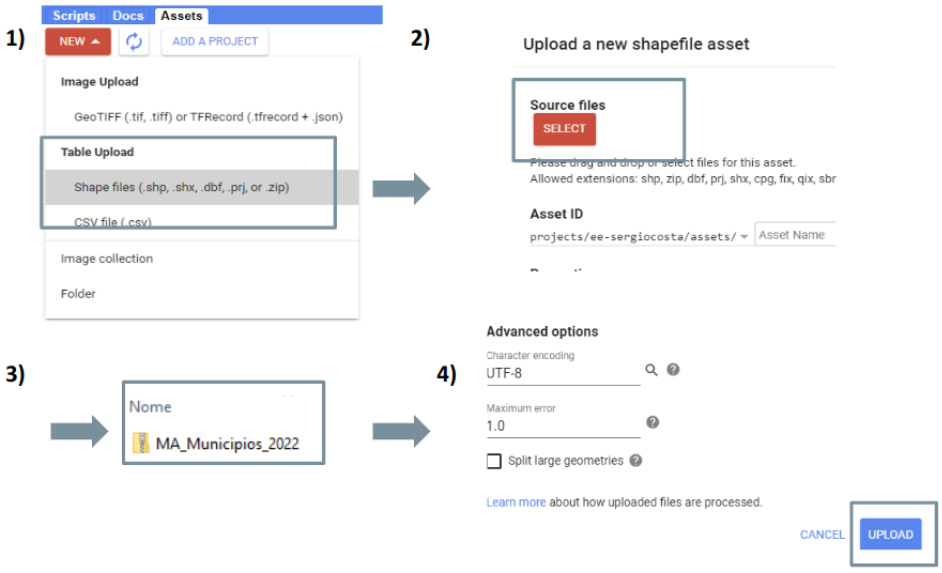


Figura 1.8. Enviando dados vetoriais

Na aba *Assets* inicia-se o processo clicando em *New* e, em seguida, escolher *Shapefiles*. Ao realizar essas etapas, uma janela será exibida para o carregamento dos arquivos. Nessa janela, clique em *Select* para escolher o arquivo que está armazenado em uma pasta local de um computador e em seguida clica-se em *Upload*. Após o arquivo ser carregado, será possível acessá-lo na aba *Assets* e copiar o seu identificador, como mostra a Figura 1.9.

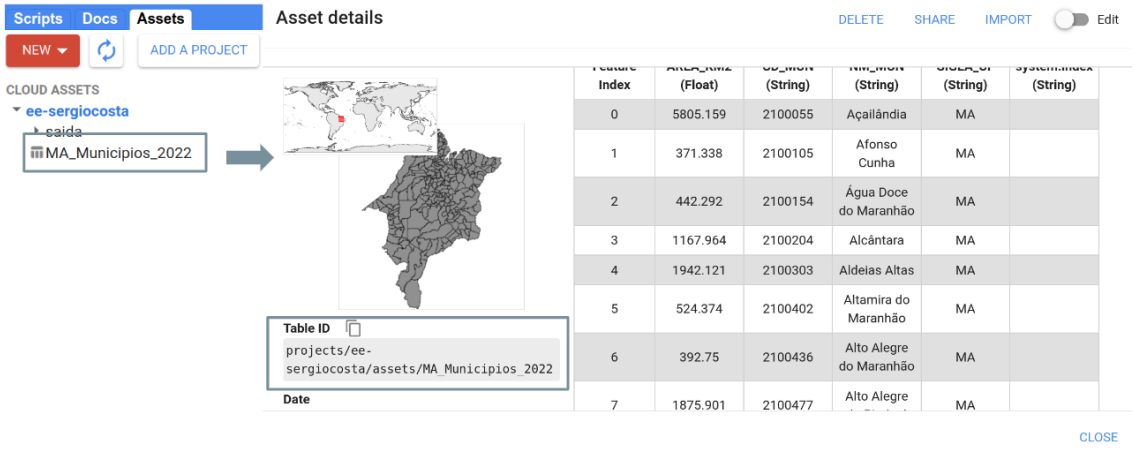


Figura 1.9. Coleções de feições enviadas para o GEEE

Com o uso desse ID, os dados podem ser carregados em um *script* no Google Earth Engine da mesma forma que foi feito com a coleção de países.

```
var municipios = ee.FeatureCollection(
    "projects/ee-sergiocosta/assets/MA_Municipios_2022"
)
```

> O formato Shapefile (também conhecido como ESRI Shapefile) é um padrão amplamente utilizado para armazenar dados geoespaciais vetoriais e tornou-se um formato comum para compartilhar, distribuir e utilizar dados geoespaciais em muitas aplicações.

1.4.1.2. Visualizando os dados vetoriais

A Seção 1.4.1.1 apresentou como acessar os dados vetoriais e associá-los a uma variável. No passo de visualização é comum adicionar os dados ao mapa com a seguinte operação `Map.addLayer(municipios)`. Porém, é possível passar alguns parâmetros adicionais, por exemplo, no código a seguir as geometrias serão apresentadas na cor vermelha e o nome **Municipios** no gerenciador:

```
Map.addLayer(municipios,{color: "FF0000"}, "Municipios")
```

Neste exemplo, a borda e o preenchimento estarão na mesma cor. Para uma definição mais detalhada, usa-se o metodo `.style` onde é possível definir diversos parâmetros,

como a cor, a largura e o estilo da linha, cor do preenchimento entre outros[1]. Por exemplo, para definir a cor da borda em preto e preenchimento em vermelho:

```
Map.addLayer(municipios.style({
  fillColor: 'FF0000', // Cor de preenchimento
  color: '000000',     // Cor da borda
  width: 2            // Largura da borda
}),{}, "Municipios")
```

No exemplo anterior todas as feições são apresentadas na mesma cor, ou seja, usou-se uma simbologia simples. Contudo, é comum a necessidade da elaboração de mapas categóricos, onde cada categoria é atribuída a uma cor, símbolo ou rótulo específico. Essas categorias podem incluir, por exemplo, tipos de uso da terra, zonas de vegetação, classes de solo, divisões administrativas, entre outros. A título de ilustração, considere um conjunto de dados contendo informações sobre as unidades federativas, onde cada unidade está associada a um atributo numérico que representa uma das cinco regiões geográficas do país. Esse código pode ser associado a diferentes cores, e para isso cria-se inicialmente uma imagem vazia:

```
var empty = ee.Image().byte();
```

A seguir, pode ser usar a função `.paint()`[2] que tem como entrada uma coleção de feições (`ufs`) e qual atributo irá ser usado definir o valor dos pixels. Neste caso, valores de 1 a 5.

```
var fills = empty.paint(ufs, 'CD_REGIAO');
```

Após a geração da imagem, é estabelecida uma paleta de cores na qual cada código está vinculado a uma cor específica. Nessa paleta, a cor "408F70" é atribuída ao valor de pixel 1, a cor "FF0000" ao valor de pixel 2 e assim por diante, seguindo a correspondência adequada..

```
var palette = ['408F70', 'FF0000', 'E7EB1B', 'FF8C4D', '9091C8'];
Map.addLayer(fills, {palette: palette, min:1, max:5}, 'colored fills');
```

O efeito pode ser observado na Figura 1.10.

1.4.1.3. Selecionando os dados vetoriais

As operações de seleção de dados vetoriais em SIGs (Sistemas de Informação Geográfica) são cruciais para analisar e extrair informações relevantes de conjuntos de dados geográficos. Essas operações possibilitam aos usuários a capacidade de filtrar, consultar e extrair elementos de dados vetoriais com base em critérios que incluem localização geográfica, atributos não espaciais, relações espaciais ou interações diretas, como, por exemplo, a seleção de objetos por meio de cliques. Da mesma forma, o Google Earth Engine (GEE) oferece a funcionalidade de realizar grande parte dessas seleções.

[1] Veja mais em https://developers.google.com/earth-engine/apidocs/ee-featurecollection-style

[2] A função `.paint` realiza uma operação conhecida como rasterização, que refere o processo de converter dados vetoriais em pixels ou pontos individuais em uma matriz (ou "raster").

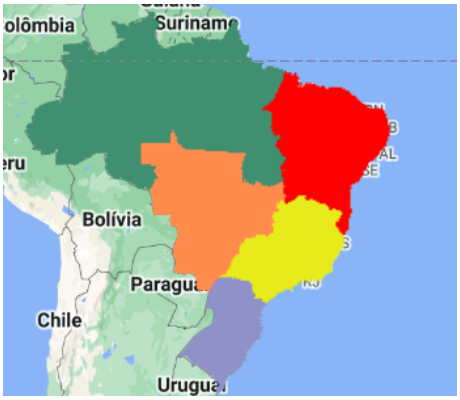


Figura 1.10. Exemplo de impressao de um mapa com cores diferentes para cada região

A **seleção por atributos** no Google Earth Engine (GEE) para dados vetoriais envolve a criação de uma expressão que filtra os elementos de uma coleção com base em atributos específicos. O GEE usa a linguagem de expressão do Earth Engine (EE) para realizar essa seleção. Considere que existe uma coleção de feições armazenadas na variável `municipios` e que agora é necessário selecionar apenas os municípios com mais de 8000 km^2. Assumindo que a área de todas as feições estão armazenadas no atributo `AREA_KM2` é possível fazer a seguinte seleção com base em atributos:

```
1 var filtroArea = ee.Filter.gt ('AREA_KM2', 8000)
2 var filtrados = municipios.filter(filtroArea)
```

A **seleção espacial** envolve o processo de filtragem de elementos com base em sua localização geográfica, como a escolha de elementos contidos em uma área de interesse delimitada por um polígono, aqueles próximos a um ponto de referência específico ou ao longo de uma linha designada. A título de exemplo, considere uma coleção contendo informações sobre todos os municípios do estado do Maranhão e outra coleção representando a delimitação da Amazônia Legal. Com esse contexto, é possível realizar a seleção apenas dos municípios que possuam pelo menos alguma porção de sua extensão geográfica inserida na área demarcada como parte da Amazônia Legal, mediante a aplicação do seguinte filtro.

```
1 var dentroAmazonia = ee.Filter.bounds(amazonialegal)
2 var selecao = municipios.filter(dentroAmazonia)
```

É possível combinar filtros, o que significa que, caso seja necessário, é possível aplicar a negação de um filtro, obtendo assim o resultado inverso.

```
1 var foraDaAmazonia = ee.Filter.bounds(amazonialegal).not()
2 var selecao = municipios.filter(foraDaAmazonia)
```

É possível combinar um filtro espacial e um filtro de atributos, como, por exemplo, para selecionar todos os municípios que não pertencem à área da Amazônia Legal e que apresentam uma área superior a 3.000 km^2:

```
1 var foraDaAmazonia = ee.Filter.bounds(amazonialegal).not()
2 var filtroArea = ee.Filter.gt ('AREA_KM2', 3000)
3 var selecao = municipios.filter(foraDaAmazonia).filter(filtroArea)
```

1.4.2. Seleção e visualização de dados matriciais

A principal fonte de dados matriciais é o catálogo do Google Earth Engine, o qual abriga coleções de imagens provenientes dos principais programas internacionais de sensoriamento remoto. A coleção Landsat se destaca por ser reconhecida como uma das fontes mais antigas e confiáveis de imagens de satélite da Terra, tornando-a uma escolha ideal para a condução de monitoramento de mudanças no uso da terra, identificação de áreas de desmatamento, análise agrícola e diversas outras aplicações. Os satélites Sentinel, que integram o programa Copernicus da União Europeia, disponibilizam uma ampla gama de dados, incluindo imagens ópticas e de radar, sendo empregados com êxito em atividades de monitoramento ambiental, detecção de incêndios florestais e acompanhamento de eventos climáticos extremos.

Adicionalmente, merecem destaque os dados provenientes do sensor MODIS (Moderate Resolution Imaging Spectroradiometer) e da missão Shuttle Radar Topography Mission (SRTM), entre outros. Além disso, é importante ressaltar a capacidade do envio de imagens provenientes de outros programas que, atualmente, não fazem parte da coleção, como é o caso do CBERS (China-Brazil Earth Resources Satellite).

1.4.2.1. Acessando os dados matriciais

No caso de imagens de satélite, é mais comum trabalhar sobre coleções de imagens ao invés de imagens individuais. Porém, similar aos dados vetoriais, é possível acessar uma dada imagem com base em seu identificador, como no código a seguir:

```
1 var image = ee.Image('LANDSAT/LC08/C01/T1_TOA/LC08_220062_20190504');
```

Sabendo como acessar uma dada imagem, a próxima seção apresenta alguns conceitos sobre visualização de dados matriciais no GEE.

1.4.2.2. Visualização de dados matriciais

A aplicação precisa dos parâmetros de visualização desempenha um papel crucial na interpretação de imagens de satélite. A título de exemplo, uma imagem de Índice de Vegetação por Diferença Normalizada gerada a partir do código a seguir possuirá valores variando de -1 a 1.

```
1 var image = ee.Image('LANDSAT/LC08/C01/T1_TOA/LC08_220062_20190504');
2 var ndvi = image.normalizedDifference(['B5', 'B4']);
```

De forma padrão, a representação de uma imagem ocorrerá em escala de cinza; contudo, é possível alterar essa configuração ao estabelecer uma paleta de cores, conforme ilustrado a seguir:

```
1 var npal = ['red', 'yellow', 'green'];
```

Utilizando essa paleta, a correspondência seria estabelecida da seguinte maneira: o valor -1 seria mapeado para o vermelho, enquanto o valor 1 seria relacionado à cor

verde. Os valores intermediários seriam alocados de forma linear nessa escala, transicionando gradualmente do vermelho para o verde e, consequentemente, pelo amarelo. Além da paleta de cores, os parâmetros para os valores mínimo e máximo têm um impacto significativo na visualização conforme demonstrado na Figura 1.11. Na imagem (a), a visualização é realizada sem a definição de uma paleta. Na imagem (b), é apresentada a visualização com a paleta definida anteriormente, porém com valores mínimos e máximos estabelecidos entre -1 e 1. Já na imagem (c), a visualização apresenta os valores mínimo e máximo ajustados para -0.5 e 0.5.

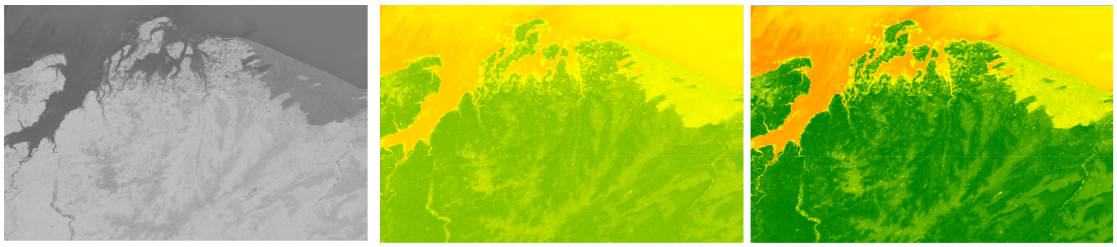


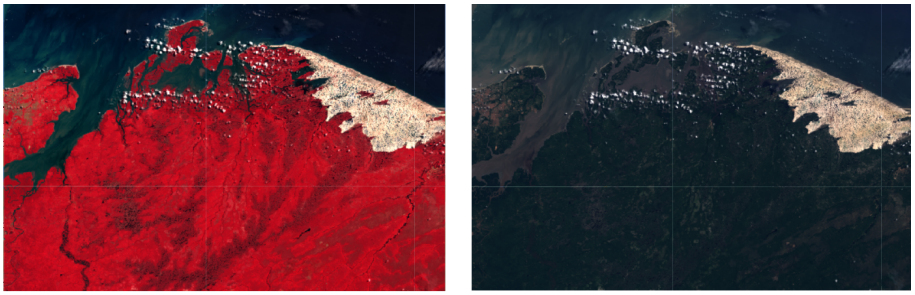
Figura 1.11. Mesma cena com diferentes parâmetros de visualização

Na interpretação de imagens, é uma prática comum a criação de composições coloridas com o propósito de destacar informações específicas. Essa técnica envolve a fusão de diferentes bandas espectrais nos canais de cores vermelho, verde e azul para gerar uma representação colorida da imagem. Nas composições em cores verdadeiras, as bandas são atribuídas aos canais correspondentes, ou seja, a banda do vermelho é vinculada ao canal vermelho, a banda do verde ao canal verde e a banda do azul ao canal azul. Nas composições em cores falsas, outras combinações de bandas são utilizadas, por exemplo, a composição B5 (infravermelho próximo), B4 (vermelho) e B3 (verde) pode ser usada para realçar a vegetação.

```
1 var cor_verdadeira = ['B4', 'B3', 'B2']
2 var falsa_cor = ['B5', 'B4', 'B3']
3 Map.addLayer(image,{bands: falsa_cor, min: 0.03, max: 0.4}, "falsa cor"
    )
4 Map.addLayer(image,{bands: cor_verdadeira, min: 0.03, max: 0.4}, "cor
    verdadeira")
```

Antes de realizar as composições coloridas é importante saber qual faixa de espectro está cada banda da imagem e qual a assinatura espectral do alvo a ser realçado. Cada alvo possui uma assinatura espectral única devido às suas propriedades físicas, químicas e estruturais, o que permite aos cientistas e analistas de sensoriamento remoto distinguir e mapear esses alvos com base nas características espectrais de suas respostas à radiação eletromagnética.

O resultado destas diferentes composições pode ser observado na Figura 1.12 que apresenta uma cena destacando os lençóis maranhenses.



a) Falsa cor b) Cor verdadeira

Figura 1.12. Exemplo de diferentes composições para uma mesma cena

1.4.2.3. Seleção de dados matriciais

Começando com um exemplo de **seleção espacial** onde deseja apenas as imagens de satélite que cobrem um dado município. Considerando que os limites do município de São Luis estão associados a variável `saoluis`, pode-se fazer a seleção como a seguir:

```
1 var imagens = ee.ImageCollection('LANDSAT/LC09/C02/T1')
2 .filterBounds(saoluis)
```

Em aplicações de sensoriamento remoto é comum a necessidade de uma **seleção temporal**, por exemplo, o código a seguir seleciona apenas as imagens de 1º de janeiro de 2022 a 30 de outubro de 2022:

```
1 var imagens = ee.ImageCollection('LANDSAT/LC09/C02/T1')
2 .filterBounds(saoluis)
3 .filterDate('2022-01-01','2022-10-30')
```

Em vez de se basear na seleção por atributos, no contexto das imagens de satélite, é uma prática comum realizar a **seleção por metadados** [1]. Esses metadados contêm informações cruciais para cada imagem, como o percentual de cobertura de nuvem. Tomando o exemplo anterior, é possível aprimorar a consulta selecionando imagens com base na cobertura de nuvens. O código a seguir demonstra como selecionar apenas as imagens com menos de 20% de cobertura de nuvens. Além disso, o resultado dessa seleção será apresentado em ordem ascendente de acordo com o nível de cobertura de nuvens.

```
1 var imagens = ee.ImageCollection('LANDSAT/LC09/C02/T1')
2 .filterBounds(saoluis)
3 .filterDate('2022-01-01','2022-10-30')
4 .filterMetadata('CLOUD_COVER', 'less_than', 20)
5 .sort('CLOUD_COVER');
```

Outra informação relevante é o par de valores “path” e “row” que é usado para indexar e localizar imagens específicas capturadas pelo sistema Landsat. O primeiro descreve a trajetória longitudinal e a segundo a latitudinal. Nesse caso, não será necessário usar o `.filterBounds` dado que esses pares ja definem a localização, como no código a seguir:

[1] Metadados são informações que descrevem outros dados, fornecendo contexto, detalhes sobre a origem, estrutura, significado e outras características dos dados principais

```
var imagens = ee.ImageCollection('LANDSAT/LC09/C02/T1')
.filterDate('2022-01-01','2022-10-30')
.filterMetadata('CLOUD_COVER', 'less_than', 20)
.filterMetadata('WRS_PATH', 'equals', 220)
.filterMetadata('WRS_ROW', 'equals', 62)
.sort('CLOUD_COVER');
```

Estes critérios são muito comuns, inclusive encontradas nos formulários de busca em catálogos de imagens de satélite. A utilização de *scripts* representa uma automação do processo que, de outra forma, exigiria que um usuário inserisse manualmente os valores em cada campo de entrada e realizasse a filtragem por meio de cliques. Além da facilidade de replicação, essa abordagem permite uma ampla variedade de combinações, que possibilitam a criação de filtros mais complexos, ficando a cargo do leitor explorar e experimentar essas possibilidades.

1.5. Processamento

Antes de avançar, é essencial adquirir uma compreensão do padrão conhecido como *filter-map-reduce*, amplamente empregado em diversas aplicações, especialmente nas que lidam com vastos conjuntos de dados. No âmbito das aplicações do Google Earth Engine, essa abordagem é uma prática comum para o processamento eficiente e escalável de dados geoespaciais, conforme exemplificado na Figura 1.13.

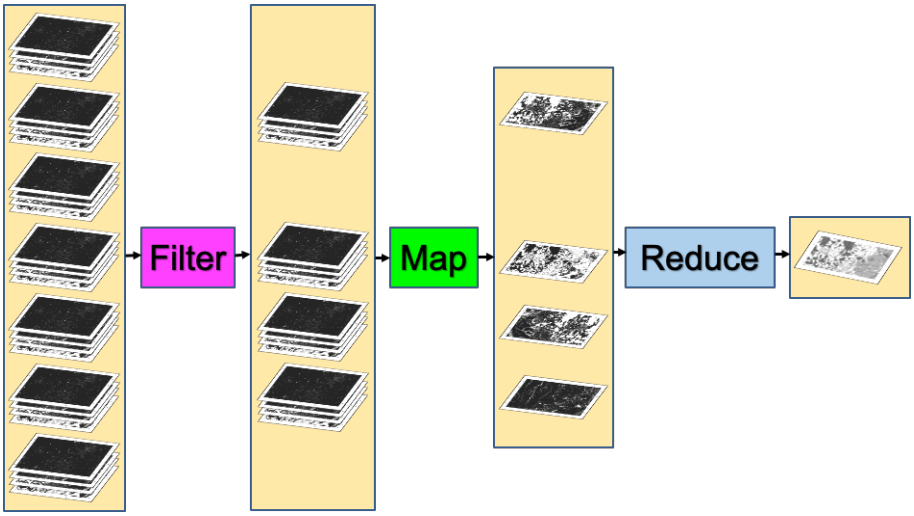


Figura 1.13. O padrão filter-map-reduce no GEE. Fonte [Cardille 2022]

A primeira etapa do processo consiste na seleção ou filtragem dos dados, como explicado na Seção 1.4. Isso é feito por meio da função `.filter()`, que permite escolher um conjunto de imagens ou feições com base em critérios espaciais, temporais, de atributos e/ou metadados. Após a conclusão da etapa de filtragem, a transição para a fase de mapeamento (*map*) ocorre por meio da utilização da função `.map()`. Nessa etapa, uma função específica é aplicada a cada elemento do conjunto de dados filtrados, independentemente de se tratar de imagens ou feições. Essa abordagem permite a realização de cálculos ou transformações nos dados. O resultado do processo de mapeamento é, por sua vez, uma coleção que pode ser combinada utilizando a função `.reduce()`.

No contexto do Google Earth Engine, essa função é frequentemente empregada para calcular estatísticas resumidas, tais como médias, somas, desvios padrão, e assim por diante, com base nos valores mapeados em cada imagem da coleção. Isso se revela valioso para a geração de mapas de resumo ou para a obtenção de estatísticas a partir de uma série temporal de imagens, como será discutido na próxima seção.

Em resumo, o padrão filter-map-reduce no contexto do Google Earth Engine permite que você selecione dados relevantes, aplique operações específicas a esses dados e, em seguida, reduza os resultados para obter informações agregadas.

1.5.1. Processando dados vetoriais

As operações sobre dados vetoriais são comumente conhecidas como operações de geoprocessamento, aqui serão apresentados alguns exemplos mais comuns.

1.5.1.1. Geração de faixas de distância

Essa operação permite identificar influência ou zonas de proximidade ao redor de entidades geográficas específicas, como pontos, linhas ou polígonos [Longley 2005]. Operações de buffer podem ser usadas em análise de proximidade, por exemplo, as autoridades municipais podem usar buffers para definir zonas de restrição ou regulamentação em torno de recursos sensíveis, como rios, reservatórios ou áreas de preservação ambiental. Para o planejamento de transporte, essa operação permite identificar áreas que estão em uma determinada distância de estações de metrô, paradas de ônibus ou estações de trem.

Para realizar essa operação, é necessário selecionar a entidade geográfica de interesse, que pode ser um ponto, uma linha ou um polígono, e definir a distância desejada em unidades geográficas, geralmente em metros. Por exemplo, imagine que desejamos investigar a possível relação entre a atividade de desmatamento e a proximidade a polos madeireiros. Suponhamos que tenhamos uma coleção de dados representando a localização dos polos madeireiros, denominada como variável `polosMadeireiros`. Podemos aplicar uma operação de criação de buffer com uma distância de 50 quilômetros utilizando o seguinte código:

```
1 var fbuffer = function (feat) { return feat.buffer(50000) }
2 var bufferPolos = polosMadeireiros.map(fbuffer)
```

1.5.1.2. Sobreposição de polígonos

A sobreposição de polígonos é uma operação fundamental em Sistemas de Informação Geográfica (SIGs) pois permitem analisar e manipular dados geoespaciais de diferentes maneiras. Essa operação é essencial para várias aplicações, como planejamento urbano, gerenciamento de recursos naturais, análise de riscos e muitos outros campos relacionados à geoinformática. As três operações principais de sobreposição de polígonos em SIGs incluem: a) intersecção, b) união e c) diferença.

A operação de intersecção é um procedimento que consiste na identificação das regiões de sobreposição entre duas ou mais geometrias. Essa operação desempenha um papel fundamental na análise de dados geoespaciais, permitindo a determinação das áreas em que diferentes conjuntos de dados compartilham elementos em comum. Por exemplo, no contexto da pesquisa sobre conflitos territoriais, a operação de intersecção é valiosa para identificar áreas onde as jurisdições de várias entidades se sobrepõem, possibilitando uma análise mais precisa e detalhada. Continuando o exemplo anterior, é possível desenvolver um *script* que identifica as áreas de terras indígenas que se encontram dentro das regiões de buffer, o que pode ser essencial para avaliar o impacto de potenciais atividades ou intervenções dentro dessas áreas.

```
var intersecao = bufferPolos.map(function(feat1) {
      return tisPA.map(function(feat2) {
            var geo = feat1.intersection(feat2);
            return ee.Feature(geo);
      });
}).flatten();
```

A operação de união consiste na combinação de polígonos, gerando um novo polígono que abrange a totalidade da área englobada pelos polígonos originais. Essa técnica é aplicada quando é necessário consolidar informações provenientes de diversas fontes ou criar um único polígono contínuo a partir de várias unidades geográficas menores. Em contextos como a gestão de recursos naturais, a união de múltiplas áreas de vegetação pode resultar em um único polígono representando uma extensa área de habitat. Como exemplo prático, considera-se a fusão de vários municípios para delinear uma determinada região de estudo. O Código 1.5 ilustra a criação de uma feição que representa a área de estudo com quatro cidades localizadas no estado do Maranhão.

```
var filtro =  ee.Filter.or(
                ee.Filter.eq('NM_MUN', 'Barra do Corda'),
                ee.Filter.eq('NM_MUN', 'Grajaú'),
                ee.Filter.eq('NM_MUN', 'Jenipapo dos Vieiras'),
                ee.Filter.eq('NM_MUN', 'Fernando Falcão')
)
var areaEstudo = ee.FeatureCollection(
    "projects/ee-sergiocosta/assets/MA_Municipios_2022")
  .filter( filtro )
  .union()

Map.addLayer(areaEstudo)
```

Código 1.5. Criando uma feição a partir da união de uma coleção de feições

A operação de diferença envolve a subtração de um polígono a partir de outro, sendo aplicada para eliminar uma área específica de um polígono maior. Por exemplo, essa técnica pode ser utilizada para remover uma área construída de um polígono que originalmente representa uma zona de conservação ambiental. Em um cenário semelhante ao apresentado no Código 1.5, suponhamos que haja uma feição representando uma área de estudo (`areaEstudo`). No entanto, durante a análise, o usuário pode desejar excluir um município específico (`grajau`). Nesse caso, é possível realizar a remoção desse município com o seguinte código:

```
var calcularDiferenca = function(feature) {
  var diferenca = grajau.map(function(feat2) {
    return feature.difference(feat2);  });
  return diferenca;
};
var resultadoDiferenca = areaEstudo.map(calcularDiferenca).flatten();
```

1.5.1.3. Cálculo de Centroides

Uma operação bastante comum envolve o cálculo do ponto central, também conhecido como centro de massa, de uma geometria, o que é denominado de centroide. Essa operação é frequentemente empregada para posicionar rótulos em mapas temáticos, realizar análises de redes, avaliar densidade e em diversas outras aplicações. Ela é aplicada à geometria de uma feição, portanto, para calcular os centroides de uma coleção, é necessário utilizar a função `.map()` para aplicá-la a toda a coleção. Por exemplo, para criar uma coleção contendo os centroides de uma coleção de municípios.

```
var calcularCentroide = function(feature) {
  var centroid = feature.geometry().centroid();
  return ee.Feature(centroid)
};
var centroides = municipios.map(calcularCentroide);
```

1.5.1.4. Gerando Dados Amostrais

A geração de amostras aleatórias é uma etapa fundamental em diversas aplicações de sensoriamento remoto e análise geoespacial. Essas amostras podem ser utilizadas para avaliar a qualidade de classificações de imagem, treinar algoritmos de aprendizado de máquina ou conduzir análises estatísticas. Utilizando o método `.randomPoints`, especificamos a região de interesse e o número de pontos aleatórios desejados. Os pontos aleatórios gerados são armazenados em uma coleção de entidades (`FeatureCollection`). No exemplo a seguir é criado uma amostra de 100 pontos em uma dada área de estudo:

```
var randomPoints = ee.FeatureCollection.randomPoints(areaEstudo, 100);
```

1.5.2. Operações sobre dados matriciais

Muitas das operações sobre dados matriciais podem ser melhor entendidas dentro do conceito de algebra de mapas [Tomlin et al. 1990]. Tomlin define as seguintes três operações:

- Operações locais: O valor de uma localização no mapa de saída é calculado a partir dos valores da mesma localização em um ou mais mapas de entrada. Elas incluem expressões lógicas, como “classificar como alto risco todas as áreas sem vegetação com inclinação superior a 15%” (Figura 1.14.a).
- Operações focais: O valor de uma localização no mapa de saída é calculado a partir dos valores da vizinhança da mesma localização no mapa de entrada. Elas incluem expressões como “calcular a média local dos valores do mapa” (Figura 1.14.b).

- Operações zonais: O valor de uma localização no mapa de saída é calculado a partir dos valores de uma vizinhança espacial da mesma localização em um mapa de entrada. Essa vizinhança é uma restrição em relação a um segundo mapa de entrada. Elas incluem expressões como "dado um mapa de cidades e um modelo digital de terreno, calcular a altitude média para cada cidade" (Figura 1.14.c).

Estas operações são ilustradas na Figura 1.14.

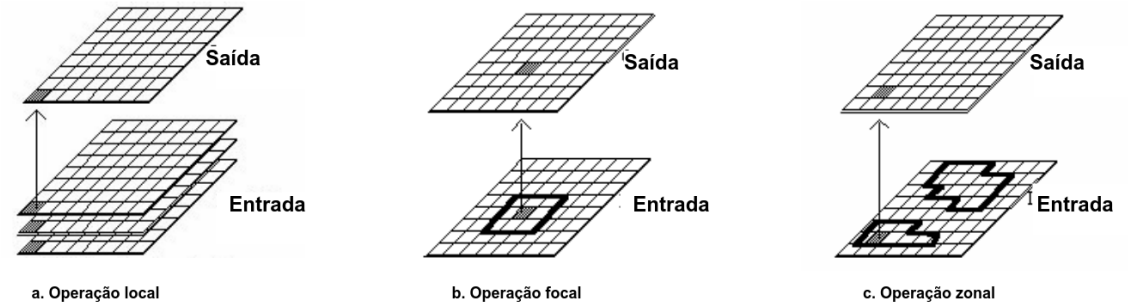


Figura 1.14. Operação da algebra de mapas de Tomlin. Adaptado de [Tomlin et al. 1990]

1.5.2.1. Operadores Locais

Operadores locais realizam operações em células ou pixels individuais em um conjunto de dados matriciais sem considerar os valores nas células vizinhas. Exemplos de operadores locais incluem operações aritméticas básicas (adição, subtração, multiplicação, divisão), bem como funções mais complexas, como logaritmos, exponenciação e operações trigonométricas. No Google Earth Engine, é possível realizar operações aritméticas básicas nas bandas ou imagens como a seguir:

```
1 var resultado = imagem1.add(imagem2);
2 var imagemRaizQuadrada = imagem.sqrt();
```

Um exemplo mais interessante é o cálculo de NDVI (*Normalized Difference Vegetation Index*). Este índice é amplamente utilizado em sensoriamento remoto para avaliar a saúde e a densidade da vegetação em uma determinada área. É um exemplo clássico de uma operação local, onde o valor do índice é calculado pixel a pixel, considerando os valores individuais de cada pixel nas bandas de um conjunto de dados de imagem. A fórmula básica para calcular o NDVI é a seguinte:

$$NDVI = \frac{(NIR - Red)}{(NIR + Red)} \tag{1}$$

Onde NIR (*Near-Infrared*) representa o valor do pixel na banda de infravermelho próximo enquanto Red representa o valor do pixel na banda vermelha. O resultado do cálculo do NDVI é um valor que varia de -1 a 1, onde valores próximos a 1 indicam

vegetação densa e saudável, valores próximos a 0 sugerem áreas com pouca ou nenhuma vegetação e valores próximos a -1 estão associados a corpos d'água ou superfícies não vegetadas. Supondo que a variável `imagem` é uma imagem landsat e as bandas 5 e 4 equivalem ao Red e NIR respectivamente, o seguinte código mostra explicitamente a aplicação da aritmética de bandas para o cálculo do NDVI.

```
var ndvi = image.expression('(NIR - Red) / (NIR + Red)', {
  'NIR': image.select('B5'), // Banda NIR
  'Red': image.select('B4')  // Banda Red
});
```

O mesmo resultado pode ser alcançado utilizando a seguinte função específica, que efetua a diferença normalizada:

```
var ndvi = image.normalizedDifference(['B5', 'B4']);
```

1.5.2.2. Operadores focais

Operadores focais, também conhecidos como operadores de vizinhança, consideram uma vizinhança específica de células ao redor de cada célula de destino ao realizar operações. Esses operadores são usados para tarefas como filtragem, suavização ou realce de características específicas nos dados. Um exemplo comum é o filtro de média, que substitui o valor de cada célula pela média dos valores de suas células vizinhas em uma vizinhança definida. Com base na imagem de NDVI criada anteriormente, é possível a geração de uma nova imagem ao aplicar a média em uma vizinhança de 3 por 3:

```
var ndvisuavizado = ndvi.reduceNeighborhood({
  reducer: ee.Reducer.mean(),
  kernel: ee.Kernel.square(3),
});
```

A Figura 1.15 destacar o efeito da operação focal em parte de uma dada cena.

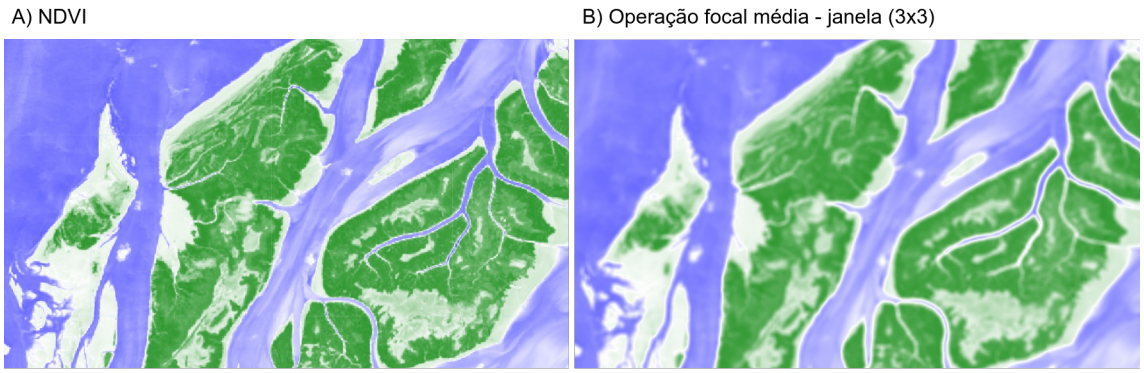


Figura 1.15. Comparando o NDVI antes e depois da aplicação da operação focal

1.5.2.3. Operadores Zonais

Operadores zonais operam em grupos de células dentro de zonas ou regiões definidas. Eles são frequentemente usados para analisar e resumir dados dentro de áreas geográficas ou limites administrativos específicos, como municípios, estados ou áreas de estudo. Operadores zonais podem calcular estatísticas como soma, média, valor máximo ou mínimo das células dentro de cada zona. Com um mapa de NDVI, previamente gerado, é possível calcular a média do NDVI para o município de São Luís:

```
var ndviSaoLuis = ndvi.reduceRegion({
  reducer: ee.Reducer.mean(),
  geometry: saoluis.geometry(),
});
```

Além destes 3 operadores, pode-se considerar ainda um quarto que são os operadores globais. Estes operadores consideram o conjunto de dados matriciais na totalidade ao realizar operações. Eles são usados para tarefas que requerem informações de todo o conjunto de dados, como equalização de histograma, estiramento de contraste ou redimensionamento de todo o conjunto de dados. No Google Earth Engine pode-se usar a função `.reduceRegion` mas sem definir uma região específica.

```
var ndvi_medio = ndvi.reduceRegion({
  reducer: ee.Reducer.mean(),
  bestEffort: true,
});
```

Observe que aqui foi usado um atributo adicional para a operação. Isso ocorreu por que neste caso o tamanho da imagem possui mais pixels do que o valor máximo esperado. Neste caso, pode-se aumentar esse valor máximo ou deixar para o Google Earth Engine decidir qual a melhor estratégia para executar essa operação. Para isso o atributo `bestEffort` foi definido para verdadeiro. Em aplicações mais complexas e com custo operacional alto, o usuário precisará analisar cada caso.

1.6. Conclusões

A abordagem de utilização da plataforma Google Earth Engine (GEE) demonstra ser uma alternativa viável e eficaz para aprimorar a acessibilidade e a eficiência no ensino de geoprocessamento e sensoriamento remoto, particularmente nas disciplinas introdutórias. O GEE proporciona acesso a uma vasta gama de algoritmos e conjuntos de dados geoespaciais, eliminando a necessidade de instalação de softwares especializados e minimizando os requisitos de recursos computacionais nos laboratórios de informática das universidades.

Essa plataforma baseada na nuvem oferece uma solução prática para a análise de fenômenos geográficos com base temporal e espacial bem definidos, abrindo oportunidades para o estudo de diversos aspectos naturais e sociais. Além disso, o GEE integra dados de diferentes fontes, como imagens Landsat, MODIS, Sentinel e dados SRTM, o que enriquece o ambiente de aprendizado e pesquisa.

Ao adotar essa abordagem, os docentes podem focar no ensino de conceitos e métodos de geoprocessamento, em vez de gastar tempo e recursos configurando laboratórios

computacionais complexos. Portanto, a utilização do Google Earth Engine como uma ferramenta educacional nas disciplinas introdutórias de geoprocessamento e sensoriamento remoto se mostra promissora e benéfica para a comunidade acadêmica.

Este texto é complementado por uma página (disponível em: https://lambdageo.github.io/minicurso-gee/) na qual é possível encontrar exemplos de códigos completos, bem como os resultados esperados em cada execução. Além disso, na página estão disponíveis links para os dados extras utilizados em alguns dos exemplos.

Referências

[Câmara 2005] Câmara, G. (2005). Representação computacional de dados geográficos. *CASANOVA, MA et al. Banco de dados geográficos. Curitiba: Mundogeo*, pages 11–52.

[Cardille 2022] Cardille, J. A. (2022). Interpreting image series. https://google-earth-engine.com/Interpreting-Image-Series/Filter-Map-Reduce/.

[DAVIS and Fonseca 2001] DAVIS, C. and Fonseca, F. (2001). Introdução aos sistemas de informação geográficos. *Belo Horizonte: Departamento de Cartografia/UFMG*.

[Ferreira 2014] Ferreira, J. (2014). *Google Apps Script: Web Application Development Essentials*. "O'Reilly Media, Inc.".

[Gorelick et al. 2017] Gorelick, N., Hancher, M., Dixon, M., Ilyushchenko, S., Thau, D., and Moore, R. (2017). Google earth engine: Planetary-scale geospatial analysis for everyone. *Remote sensing of Environment*, 202:18–27.

[Longley 2005] Longley, P. (2005). *Geographic information systems and science*. John Wiley & Sons.

[MapBiomas 2021] MapBiomas (2021). Mapbiomas project-collection 6.0 of the annual series of land use and land cover maps of brazil.

[Smith et al. 2007] Smith, M. J., Goodchild, M. F., and Longley, P. A. (2007). *Geospatial Analysis: A Comprehensive Guide to Principles, Techniques and Software Tools*. Troubador Publishing, 2nd edition.

[Tomlin et al. 1990] Tomlin, C. D. et al. (1990). *Geographic information systems and cartographic modeling*, volume 249. Prentice Hall Englewood Cliffs, NJ.

[Wirfs-Brock and Eich 2020] Wirfs-Brock, A. and Eich, B. (2020). Javascript: the first 20 years. *Proceedings of the ACM on Programming Languages*, 4(HOPL):1–189.