Geoprocessamento no Google Engine

Guia *rápido* de uso

Guilherme de Castro Oliveira

M. Sc. Solos e Nutrição de Plantas

Eng. Florestal – UFV

O que é Earth Engine? – Fazer uma breve introdução da plataforma

- Plataforma de análise e visualização de ‘big data’
- Desenvolvido para cientistas, não engenheiros de software!
- Possui mais de 300 coleções de dados prontas para uso
- Gratuito

Conhecimentos adquiridos ao longo do tutorial (colocar os que faltam)

Script 1 - Criando um polígono com área de interesse e carregando uma coleção com melhores pixels da área

- Criar uma geometria da área de interesse
- Carregar uma coleção de imagens
- Filtrar imagens da coleção com geometria da área de interesse
- Filtrar imagens da coleção por data
- Filtrar imagens da coleção por cobertura de nuvens.
- Reduzir uma coleção de imagens para uma única imagem
- Configurar parâmetros de visualização
- Adicionar parâmetros de visualização no código
- Salvar o script
- Carregar um script salvo

Script 2 – Criando um mapa de NDVI com imagens Landsat 8

- Outro método para compor mosaico Landsat 8 sem nuvens
- Seleção de bandas
- Álgebra de Mapas
- Métodos para cálculo do NDVI

Script 3 – Série temporal NDVI, exportação de imagens

- Criar uma função
- Aplicar a função à coleção de imagens
- Renomear uma banda
- Filtrar por metadados – cobertura de nuvens na cena
- Criar gráficos de séries temporais
- Configurar parâmetros dos gráficos
- Usar uma paleta de cores
- Exportar imagem

Script 4 – Classificação supervisionada de imagens

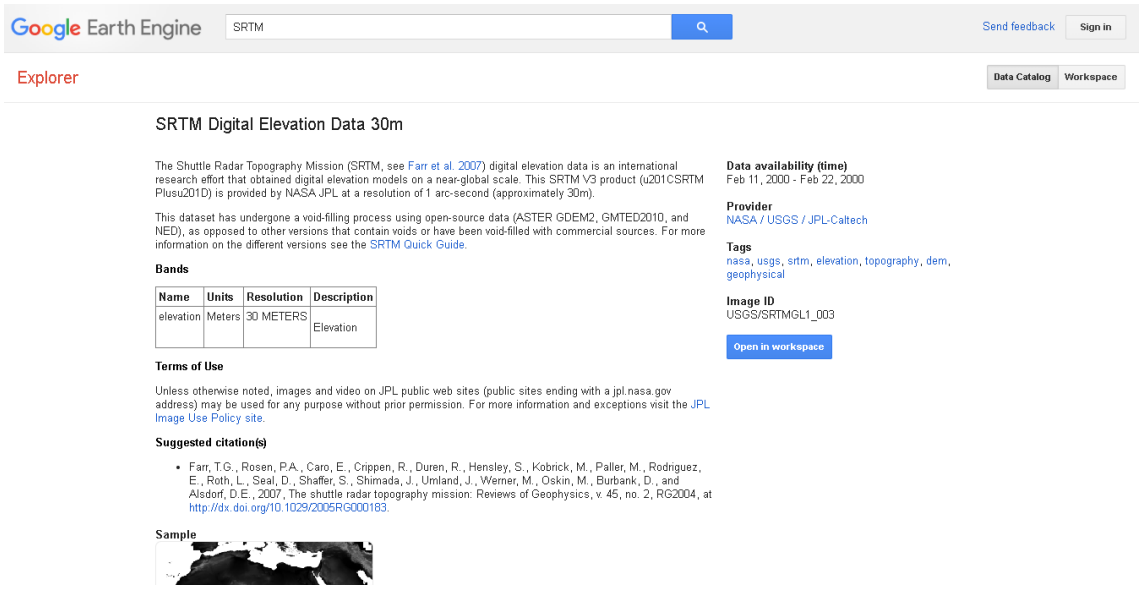
- Criar uma função
-

Plataformas do App Engine

O App Engine disponibiliza dados de inúmeros sensores remotos e permite o processamento desses dados através de linguagens de programação de alto nível. Há duas plataformas no sistema, uma de visualização de dados (*explorer*) e outra onde é feita a programação (*code editor*).

Primeiramente vamos conhecer a plataforma *explorer*, para visualizar algumas bases de dados disponíveis no sistema. Acesse o endereço https://explorer.earthengine.google.com

Nesse site é possível visualizar rapidamente as imagens e seus metadados. Por exemplo, no campo de busca, digite "SRTM". Na lista que se abrirá, posicione o cursor sobre "SRTM Digital Elevation Data 30m" e clique em "details...". A página exibida na figura abaixo será acessada, onde estão listados os metadados do sensor ou produto escolhido. Nós acessaremos frequentemente esses metadados durante a programação, pois por eles saberemos o código para identificar da coleção, o nome das bandas dos sensores e quais informações elas trazem.

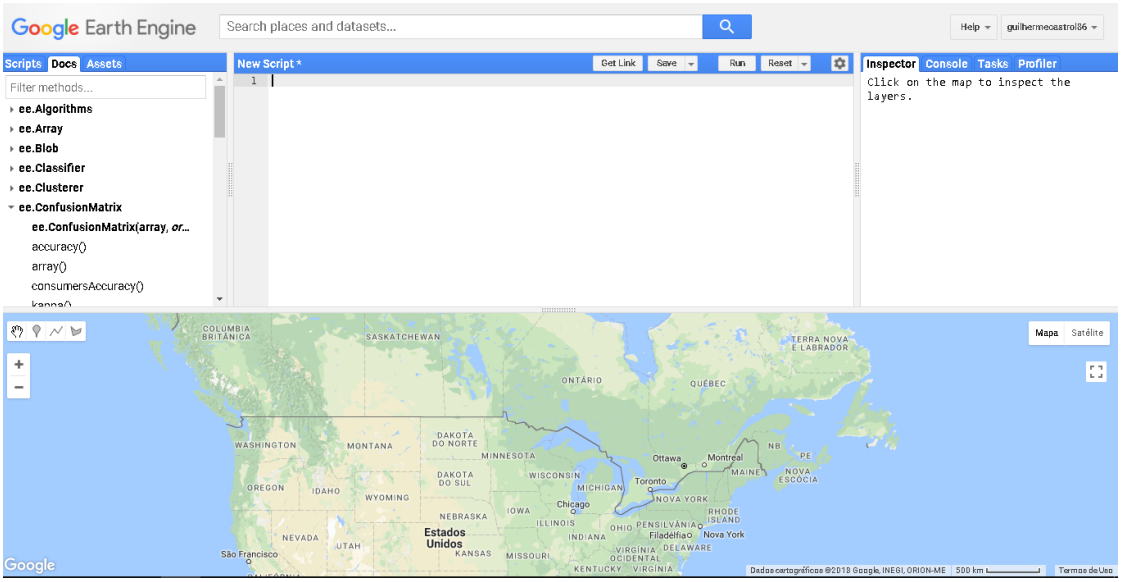


Clique em "Open in workspace" para carregar os dados e exibi-los no mapa. Rapidamente os dados serão exibidos em escala continental ou mesmo global. Da mesma forma que fizemos para SRTM, podemos acessar qualquer base de dados presente na plataforma. Experimente acessar outras bases de dados disponíveis!

Na plataforma *Code Editor*, os dados também podem ser acessados, porém sua visualização não é tão simples quanto na plataforma *Explorer*.

A plataforma de código pode ser acessada pelo link https://code.earthengine.google.com (é necessário ter uma conta ativa do Google para acessar o editor de códigos). Na página exibida na figura abaixo há 4 seções, cada qual com uma função.

À esquerda, temos a janela de scripts e documentação dos métodos, onde estão armazenadas diversas funções ou mesmo scripts completos para executar uma determinada tarefa. Procure, por exemplo, 'NDVI' na aba *'Scripts'* Veja que há diferentes scripts que utilizam o NDVI para alguma finalidade, ou que servem para representá-lo em grandes escalas. Na aba '*Docs'* estão disponíveis várias funções (métodos) incorporadas ao Earth Engine. A aba '*Assets'* acessa o gerenciador de arquivos. É onde ficam armazenados os dados enviados pelo usuário via *upload* para o sistema.



Escolha o script "Normalized difference" e veja que o mesmo será adicionado na janela do código. Observe que há 6 declarações no script para executar a tarefa. Mais adiante isso será abordado com mais detalhes. Na janela do código, além de escrever as declarações do mesmo, estão as opções de salvar o código (*Save*) de reiniciá-lo (*Reset*) e de apaga-lo (*Clear script*).

Na janela da direita, temos as abas *Inspetor, Console* e *Tasks*. O inspetor é usado para visualizar informações das camadas que foram adicionadas no mapa, como o nome do sensor, nome da banda, valor numérico do pixel, campos da tabela de atributo, etc. O console serve para exibir resultados de funções executadas, mensagens criadas pelo usuário ou mensagens de erro. *Tasks* (tarefas) é utilizado quando o cliente está baixando ou subindo dados para a plataforma.

A última janela é o *map*, onde são exibidos as camadas e os mapas. Clique em "*Run*" para executar o script. Veja que a imagem NDVI aparece no mapa. A imagem MODIS que

foi utilizada para seu cálculo também foi adicionada, porém está abaixo da anterior. Para alternar entre uma imagem e outra ou alterar suas configurações de exibição, posicione o cursor sobre o botão "*Layers*". Perceba que há duas camadas: "NDVI" e "MODIS bands 1/4/3". A opacidade ou transparência de cada camada é configurada no botão deslizante à direita do seu nome. No botão parâmetros de visualização (símbolo da engrenagem), podemos escolher a composição de bandas de cada camada (no caso de imagens tipo *raster*), a função de distribuição dos dados, paleta de cores, etc.

Outra função importante da janela de exibição é a criação manual de geometrias, as quais são importadas no código de forma automática. Podemos criar geometrias dos tipos ponto, linha e polígono. Clique no ícone "Desenhar forma", no canto superior esquerdo da janela de exibição. Perceba que, ao clicar no ícone, uma nova entrada aparece no código, para importação da geometria a ser desenhada. Desenhe um polígono, para fechá-lo basta clicar no primeiro vértice uma única vez. Novas geometrias (feições) podem ser adicionadas à essa mesma camada, bastando repetir o processo da mesma forma.

Para criar geometrias em uma nova camada, selecione o botão "Geometry imports" e clique em "+ new layer". Atenção, pois as geometrias serão armazenadas na camada que foi selecionada nessa lista!

Agora, veremos a estrutura de sintaxe da linguagem Java, que é a utilizada para escrever códigos no App Engine.

Sintaxe JavaScript no Earth Engine

A seguir, serão mostrados alguns exemplos de comandos básicos. Essa mesma estrutura de sintaxe será aplicada posteriormente nos códigos. Abra um novo código em branco para executar cada uma das declarações abaixo. Para executar, clique em "*Run*" ou pressione ctrl+Enter.

a) Para **imprimir** alguma coisa no console: função print(). Variáveis do tipo texto (*strings*) devem ser colocadas entre aspas:

```
print('Hello World!')
```

b) Para **comentar** o código, use duas barras '/'. Repare que o comentário ficará na cor verde. Para comentar uma **seção** do código, utilize "/*" no início do comentário e "*/" no fim do comentário.

```
// print('Hello World!')

/* print('Hello World!') */
```

c) Para criar uma **nova variável**, use a palavra reservada 'var' com a seguinte sintaxe: var nome = valor

```
var oi = 'ola'      // variável string, deve estar entre aspas
var num = 400           // variável tipo numérica não usa aspas
```

d) Para **imprimir o valor** contido numa variável:

```
print(oi)          // vai imprimir 'ola'
```

e) Também podemos **concatenar** o valor da variável com algum texto explicativo, separando os elementos com uma vírgula ou com operador de soma (+).

```
print('valor:' , num)   //imprime 'valor: 400',linhas separadas
print('valor:' + num)   //imprime 'valor: 400', na mesma linha
```

Nesse caso, podem ser adicionados à função print tantos argumentos quanto quisermos. Observe que cada um desses argumentos será impresso **numa linha diferente** do console quando usarmos vírgula como separador, e são impressos na **mesma linha** quando usamos "+".

d) Para criar uma variável do tipo **lista**, use colchetes:

```
var elementos = [30, 22, 'c', 'd']
```

Veja que numa lista podemos misturar diferentes tipos de objetos. No exemplo acima, uma lista contém números e texto. Poderíamos também incluir funções, dicionários (veremos a seguir) etc.

Para retornar um único elemento de uma lista, usa-se colchetes após o nome da variável, contendo o índice do elemento. A contagem começa no 0, portanto o primeiro elemento da lista possui índice 0, o segundo possui índice 1 e assim sucessivamente.

```
print(elementos[0])     // imprime '30'
print(elementos[3])     // imprime 'd'
```

e) **Dicionário** é um objeto no JavaScript, que armazena pares de chaves (*key*) e valores (*value*). São muito usados para filtrar metadados de imagens de satélite e para passar argumentos para funções. No Google Engine iremos lidar frequentemente com esse tipo de estrutura para selecionar bandas, datas de aquisição, etc.
Dicionários são criados usando **chaves** '{', para declarar seu conteúdo. Cada chave é seguida por **dois pontos** ':' e o seu respectivo valor é declarado após os dois pontos. O valor pode ser numérico, texto, lista ou mesmo outro dicionário. As chaves são

separadas por **vírgula** ‘,’. Vamos criar um dicionário chamado ‘metadata’ e, em seguida, imprimir seus valores no console.

```
var metadata = {
  sensor: 'OLI',                  // string
  resolution: 30,                 // numeric
  view: ['nadir', 'off-nadir']    // list
}
print(metadata)
```

Veja que o dicionário possui três chaves (*resolution*, *sensor* e *view*) e cada chave possui seu valor. Os **itens de um dicionário** podem ser acessados, ou impressos, de duas formas diferentes:

```
print(metadata['sensor'])    // Usando colchetes c/ nome do item
print(metadata.sensor)       // Usando um ponto
```

Nos casos em que o valor é uma lista, como na chave *view* do dicionário ‘metadata’, para acessar algum elemento da mesma podemos usar o seu índice, expresso em colchetes, com a seguinte sintaxe: dicionario.item[índice], lembrando que o primeiro item tem índice 0. No exemplo em questão, a chave ‘view’ contém uma lista com 2 itens. Vejamos:

```
print(metadata.view[0])
```

Essa é a mesma sintaxe que usamos para obter elementos de uma lista!

f) **Funções** são operações e declarações agrupadas para facilitar a leitura e o reuso do código. São extremamente úteis na programação! Para construir uma função, é usada a palavra reservada ‘***function’***. Uma função deve conter argumentos (parâmetros) e uma saída (*output*), que é determinada pelo comando ‘***return’***. O conjunto de declarações que compõem uma função são colocadas entre chaves. A seguir temos o exemplo da estrutura e sintaxe de uma função.

```
var funcao = function(parametro1,parametro2, ... ) {
  declaração;
  declaração;
  return resultado;
};
```

Vamos a um exemplo prático. Criaremos uma função que calcula a área de um círculo de raio a ser definido pelo usuário. Precisamos então que nossa função tome o valor do raio, eleve ao quadrado e multiplique por pi (3,14). Esse valor será armazenado numa nova variável que será a saída da nossa função.

```
var areaCirculo = function(raio){
```

```
        var area = raio*raio*3.14;          //identação*
        return area
}
```

A primeira linha cria uma função e associa a mesma à variável 'areaCirculo'. Esse é o nome da função, que será usado para chamá-la posteriormente no código. O valor contido nos parênteses é o argumento da função, podem ser tantos argumentos quantos forem necessários, até mesmo nenhum. As funções onde as variáveis são declaradas fora da mesma são denominadas 'globais'.
Uma vez criada a função 'areaCirculo', vamos usá-la para calcular a área de um círculo com 10 metros de raio.

```
print(areaCirculo(10))  // imprime a área do círculo
```

Vamos melhorar a exibição do resultado, adicionando informações para torna-la mais inteligível:

```
print("A área do círculo é de "+ areaCirculo(10)+ "m²")
```

Observação: A linguagem Java não exige a identação das declarações contidas numa função ou em um loop, porém é recomendável fazê-la para facilitar a leitura do código.

Além das funções que nós mesmos podemos definir, há várias delas já incorporadas no App Engine!

Objetos Earth Engine

Até o momento, todas variáveis, objetos e funções que criamos estão apenas armazenadas no ambiente do usuário (cliente). Para que esses elementos sejam carregados para o servidor, são utilizados métodos denominados **construtores** para armazená-los em '*containers*'.

Por exemplo, a função **ee.Number()** toma um número e atribui o mesmo a uma variável armazenada no servidor. Para ilustrar essa diferença, façamos um teste. Primeiro, vamos criar uma variável numérica no cliente (usuário) e imprimir seu valor. Depois, vamos salvar essa variável no servidor e imprimir seu valor.

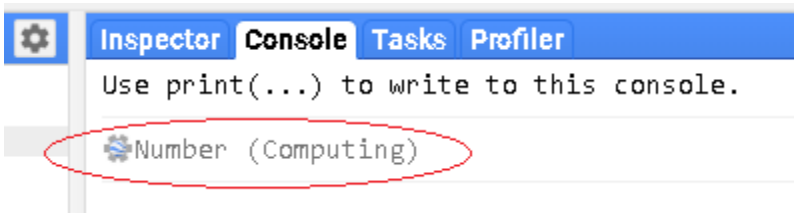
```
var num = 300                   // Cria uma variável no cliente
print(num)                          //imprime valor de num
```

O processamento é instantâneo. Isso por que não utilizamos o servidor. Agora, iremos salvar essa variável em um container no servidor e depois imprimir seu valor:

```
var num = ee.Number(300)     // Cria uma variável no servidor
```

```
print(num)
```

Perceba que o processamento demorou mais que o esperado, para uma operação tão simples quanto criar uma variável numérica e imprimir seu valor. É possível ver no console que o servidor está processando os dados. Esse maior tempo de processamento se deve ao fato que os dados são enviados para o servidor, armazenados e depois enviados de volta para o usuário.



Todas as variáveis que utilizamos nos códigos devem ser salvas em containers através dos construtores. Caso contrário, o servidor pode não ser capaz de reconhecer essas variáveis, causando erros de processamento.

Quando um parâmetro é armazenado no servidor, é criado um objeto Earth Engine. Esses objetos não podem mais ser processados com funções comuns do JavaScript (*e.g* Math.sqrt()). Agora, serão utilizados **métodos** próprios para processamento de objetos Earth Engine.

Por exemplo, para calcular a raiz quadrada de um número no usuário:

```
var num = 300                 // variável no cliente
var raiz = Math.sqrt(num)               // função Math.sqrt()
print(raiz)
```

Para calcular o mesmo valor de um número no servidor, a função Math.sqrt() não se aplica:

```
var num = ee.Number(300) // variável no servidor (container)
var raiz = Math.sqrt(num)         // não funciona!
print(raiz)                       // retorna “NaN”
```

Devemos, nesse caso, usar o método sqrt() aplicado ao objeto ‘num’:

```
var raiz = num.sqrt()
print(raiz)
```

A mesma lógica é usada para listas e dicionários. Vejamos alguns exemplos.

```
var lista = ee.List([1, 2, 3, 4, 5]); // cria lista no servidor
lista.get(2) // método get() para retornar valor de índice 2
```

Veja que em vez de lista[2], que seria aplicado à uma lista comum para retornar o valor de índice 2 da lista, usamos o método **get()** no objeto Earth Engine 'lista' para retornar o valor de índice 2.

Para dicionários, o construtor utilizado é **ee.Dictionary()**:

```
var metadata = ee.Dictionary({
  sensor: 'OLI',                  // string
  resolution: 30,                 // numeric
  view: ['nadir', 'off-nadir']    // list
})

print(metadata.get('sensor'))     // use get() para obter valor
print(metadata.keys())            // use keys() para todas chaves
```

Datas

Um tipo de objeto muito importante no Earth Engine são as datas. Data é um tipo especial de dados, armazenados com o construtor **ee.Date**. Frequentemente filtramos as séries de dados que utilizamos pelas datas, determinando o início e o fim de uma série ou selecionando uma data específica em uma série de observações.

Da mesma forma que os objetos vistos anteriormente, os objetos *Date* se distinguem quando armazenados no cliente ou no servidor. Um objeto *Date* pode ser construído a partir de uma *string* ou a partir de um objeto JavaScrip *Date*. Vejamos:

```
var date = new Date('2015-12-31') // objeto date no cliente
var date = ee.Date(date)     // objeto date no servidor
```

A seguir, são apresentadas algumas configurações do construtor **ee.Date** para diferentes formatos de datas.

```
var time = new Date('2018-12-31') // armazenada no cliente
print(time)                       // "2018-12-3100:00:00"

var eeTime = ee.Date(time)        // armazenada no container
print(eeTime)                 // "Date (2018-12-31 00:00:00)"
```

Também podemos criar um container diretamente, sem a necessidade de criar previamente um objeto JavaScript *date*. Podemos fazer isso simplesmente declarando a data no formato *string* ('2017-1-13') ou usando o método **fromYMD()**. Esse método é usado para passar os parâmetros para o construtor ee.Date separadamente. Os argumentos da função são, respectivamente: ano, mês e dia. Há

vários outros métodos que podem ser aplicados juntamente ao construtor ee.Date (busque '*date*' na aba Docs), muito úteis para trabalhar com séries temporais, por exemplo.

```
var aDate = ee.Date('2017-1-13')            // usando strings
var aDate = ee.Date.fromYMD(2017, 1, 13); // método .fromYMD()
```

Para selecionar uma data ou um período específico de uma série de dados, usaremos o método **filterDate()** aplicados a containers de **coleções de imagens**. Por exemplo, considere uma coleção de imagens Landsat 8 OLI. Essa coleção inclui todas as imagens obtidas pelo sensor desde o início de sua operação. Por padrão, quando carregamos uma coleção de imagens no Earth Engine, ela é carregada por completo, exceto se especificarmos uma data ou período específico.

Uma maneira de filtrar as coleções de imagens por **datas** é:

a) criar um objeto *date* com a data de início

b) criar um objeto *date* com a data de término

c) carregar a coleção de imagens aplicando o filtro de datas.

Esse formato é interessante quando utilizamos funções que tomam datas como argumentos. Para não termos que alterar a função quando alteramos as datas, é mais fácil defini-las como variáveis fora da função.

Obviamente, para uma única data, somente um objeto *date* será criado. Vejamos, na prática, como isso é feito. Iremos carregar imagens da série Landsat 8 e filtrá-la, deixando somente observações entre dezembro de 2015 e dezembro e 2017. Depois, adicionaremos no mapa o resultado.

```
var inicio = ee.Date('2015-12-01');    // data de início
var fim = ee.Date('2017-12-01'); // data de término

// carregar coleção
var landsat = ee.ImageCollection('LANDSAT/LC08/C01/T1')
// aplicar filtro com data de início e fim
         .filterDate(inicio, fim)
// adicionar camada no mapa
Map.addLayer(landsat)
```

Aqui utilizamos três operações novas. Depois de definir as datas nos containers, usamos o construtor **ee.ImageCollection**. Esse construtor é usado para carregar as coleções de imagens disponíveis no servidor do Earth Engine. Para isso, basta passar

para o construtor o código (ID) da base de dados. Esse código é encontrado nos **metadados** da coleção (visto na primeira seção do tutorial). Depois, aplicamos o filtro com o método **filterDate()**, declarando as datas de início e fim. Por fim, usamos a função **Map.addLayer()** para carregar as imagens no mapa. Essa função é usada para carregar **qualquer tipo** de camada e apresentá-la no mapa.

Da mesma forma que aplicamos um filtro de datas na nossa coleção, podemos também selecionar somente uma órbita-ponto, somente algumas bandas, ou somente determinados valores dentro de uma banda, etc. Para cada tipo de filtro há um método específico. Essas operações serão abordadas mais adiante.

Programando no Earth Engine

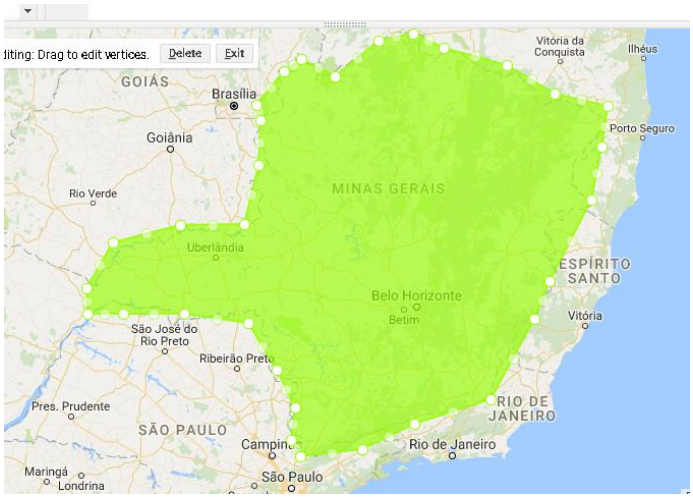
Agora que estamos familiarizados com as plataformas do Earth Engine e com a sintaxe básica do JavaScript, começaremos de fato a aplicar o conhecimento para escrever códigos. Iremos começar com códigos mais simples e, gradativamente, adicionaremos mais elementos para executar tarefas cada vez mais complexas, utilizando funções definidas pelo usuário, loops, etc. Usaremos exemplos bastante próximos da nossa realidade, com tarefas que são feitas rotineiramente no universo do geoprocessamento e da análise ambiental.

Script 1 - Criando um polígono com área de interesse e carregando uma coleção com melhores pixels da área

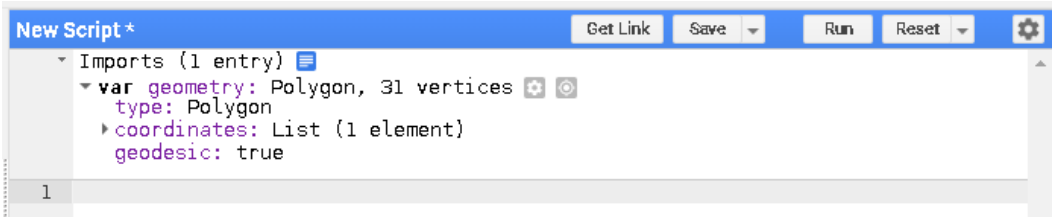
Um produto muito desejado pelos usuários é o mosaico de imagens livre de nuvens. Esses mosaicos só podem ser criados quando consideramos várias observações de um mesmo local, para obter os melhores pixels da coleção de imagens, uma vez que é muito raro encontrar uma cena completamente livre de nuvens. No geoprocessamento convencional, essa prática é demasiado custosa, pois demanda baixar todas as imagens do provedor, armazená-las e depois processar um grande volume de dados para criar o mosaico.

Numa perspectiva otimista, para criar um mosaico Landsat OLI para o Estado de Minas Gerais, por exemplo, um analista levaria dias ou até semanas para chegar ao produto final. Iremos, a seguir, criar um mosaico livre de nuvens de uma coleção Landsat 8 OLI em questão de **segundos**. No exemplo a seguir, usaremos uma técnica na qual o mosaico é composto pelo valor **mediano** dos pixels da série, após eliminar cenas com muitas nuvens.

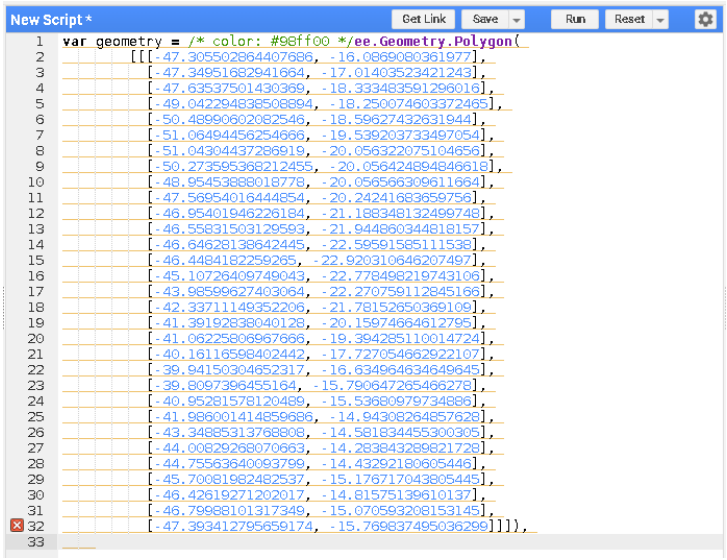
Primeiro, criaremos um polígono manualmente. Esse polígono irá conter os limites do estado de MG. O método para criar polígonos manualmente foi explicada na primeira seção do tutorial. Após a edição, teremos um polígono como o da figura abaixo.



Perceba que após finalizar o polígono, o Earth Engine automaticamente incorpora no código uma entrada para o mesmo numa seção à parte. Mas, essa geometria ainda não foi carregada no código, somente armazenada como um objeto **no servidor**. Para chamar a geometria numa função, basta usar o seu nome.



Essa é uma funcionalidade muito útil do Earth Engine, que contribui para deixar o código mais legível. Todas coleções de imagens e geometrias, quando carregadas, podem ser alocadas na seção "*Imports*" e, posteriormente, chamadas no código por construtores. Para fins de comparação, veja na imagem abaixo a importação da mesma geometria declarada diretamente no código.



Vamos alterar o nome desse polígono de 'geometry' para 'mg'. Basta clicar no nome da variável, na seção de importação, e escrever o novo nome. Em seguida, iremos carregar esse polígono no mapa.

```
Map.addLayer(mg)
```

Dica: o editor de códigos possui a função de autocompletar o código. Pressione ***ctrl+espaço*** *para autocompletar. Por exemplo, escreva "Map." e use a função autocompletar. A documentação das funções também pode ser visualizada nessa janela.*

Agora, carregaremos uma coleção de imagens Landsat 8 (ID 'LANDSAT/LC08/C01/T1_TOA'), filtrando somente as imagens da coleção que foram adquiridas entre abril de 2015 e junho de 2018. Usaremos também o polígono que criamos para selecionar cenas que **intersectam** a área, com o método **filterBounds()**.

```
Map.addLayer(mg)
var collection = ee.ImageCollection('LANDSAT/LC08/C01/T1_TOA')
    .filterBounds(mg)
    .filterDate('2015-04-01', '2018-07-01')
Map.addLayer(collection)
print(collection)
```

Carregamos, assim, uma coleção de imagens que abrange a área de interesse, no período especificado. As informações da coleção foram impressas com a função **print()**. Para ver essas informações, abra o console. Veja o número de imagens presentes, o número de bandas, as informações sobre as bandas, etc. Temos um número muito grande de imagens da área de interesse. Porém, para nós, uma única cena de cada órbita-ponto é suficiente para compor o mosaico.

Mas, como selecionar a melhor cena? Um parâmetro comum é o percentual de cobertura de nuvens. O que faremos então é filtrar toda a coleção de imagens quanto a nebulosidade. Selecionaremos somente aquelas onde o percentual de cobertura de nuvens é menor que 2%. Para a coleção Landsat 8, acessamos esses metadados da cena através da string 'CLOUD_COVER' (esse nome é diferente para cada coleção). O método para **filtrar** a coleção a partir de um valor nos metadados é **filterMetadata()**.

Adicionando essa função, o código ficará assim:

```
Map.addLayer(mg);
var collection = ee.ImageCollection('LANDSAT/LC08/C01/T1_TOA')
    .filterBounds(mg)
    .filterDate('2015-04-01', '2018-07-01')
    .filterMetadata('CLOUD_COVER', 'less_than',0.2)

Map.addLayer(collection,{bands: ['B3', 'B3', 'B2']},'landsat 8')
print(collection)
```

Além do filtro de nuvens, adicionamos dois novos argumentos na função Map.addLayer. O primeiro argumento é um dicionário, com uma chave e uma lista de valores. Através dele, selecionamos as bandas que **serão exibidas** no mapa. Note que as outras bandas ainda estão presentes na coleção, porém não são exibidas. O segundo argumento adicionado é uma *string* com o nome que será dado à camada que está sendo adicionada: 'landsat 8'. É importante dar um nome de fácil reconhecimento para as camadas, caso contrário a diferenciação se torna complicada na edição das simbologias e exibição do mapa.

O Earth Engine não define automaticamente bons **parâmetros de visualização** para imagens. Além disso, precisamos também indicar qual a composição de bandas queremos. No exemplo em questão, as imagens estão muito escuras em razão da configuração de visualização da camada e das bandas exibidas. Para alterar as configurações, acesse o botão "Layers" na janela do mapa e abra os parâmetros da camada 'landsat 8'. Selecione as bandas B4 (*red*), B3 (*green*) e B2 (*blue*), nessa ordem. Na opção "Range", selecione "Stretch 2" e aplique as configurações.

Para automatizar esse processo, podemos importar essa configuração e passa-la para o método addLayer(). Abra novamente nos parâmetros de visualização da camada 'landsat 8', clique em "Import". Perceba que a uma nova entrada, denominada '**imageVisParam**' aparece na seção 'Import' do editor de códigos.

Esse objeto guarda todas as configurações de visualização que estão definidas para o mapa atual, incluindo as bandas, valor mínimo e máximo da exibição, gama e

opacidade. Basta agora passar essa variável para a função addLayer para que esses parâmetros sejam definidos automaticamente ao carregar o mapa. Portanto, a declaração para carregar as imagens no mapa ficará dessa forma:

```
Map.addLayer(collection,imageVisParam,'landsat 8')
```

A última tarefa do nosso primeiro código será cortar (‘*clip*’) as cenas para o polígono que representa nossa área de interesse. Porém, até o momento o que temos é uma **coleção** de imagens e o método .clip() só pode ser aplicado a uma **única** imagem. Podemos solucionar essa questão de duas formas.

a) usando um iterador para aplicar a função em cada imagem da coleção, ou

b) reduzindo todas as imagens a uma única imagem, usando alguma estatística (e.g média, mediana, mínimo, etc)

Como ainda não abordamos os iteradores, optaremos pela segunda alternativa. Serão necessárias somente duas operações: uma para retornar os pixels de valor mediano da coleção e outra para cortar essa imagem para a área de interesse.

A mediana é calculada com o método **median()** aplicado à coleção de imagens. Esse redutor calcula a mediana de todos os valores de um mesmo pixel em diferentes cenas. Em seguida, essa imagem será recortada com o método **clip()**. A seguir é apresentada uma maneira de realizar essa operação, após a importação da coleção.

```
var median = collection.median()
    .clip(mg);
```

Poderíamos também fazer essas duas operações diretamente na importação da coleção, obtendo o mesmo resultado:

```
var collection = ee.ImageCollection('LANDSAT/LC08/C01/T1_TOA')
    .filterBounds(mg)
    .filterDate('2015-04-01', '2018-07-01')
    .filterMetadata('CLOUD_COVER', 'less_than',2)
    .median()      // redutor
    .clip(mg)
```

Traduzindo o código, o que declaramos foi: carregue a coleção de imagens Landsat 8, filtre apenas imagens que intersectam a geometria ‘mg’, filtre cenas obtidas entre 2015 e 2018, filtre cenas com cobertura de nuvem menor que 2%, calcule o valor mediano e corte as imagens com o polígono ‘mg’.

Parabéns! Você acaba concluir seu primeiro código em Earth Engine!

É importante **salvar** o script. Ele será armazenado na sua conta no Google Engine. Clique em '*Save*'. Nomeie o script como 'script1'. No campo '*description*', é importante deixar alguma informação para saber posteriormente qual é a função do script. Podemos escrever, por exemplo: 'cria mosaico Landsat8 para área de estudo'. Após salvar o código, ele ficará armazenado na aba '*Scripts*', na seção '*Owner*', a partir de onde poderá ser **carregado** posteriormente. Repare que a seção '*imports*' do script também é salva.

SCRIPT 1

```
Map.addLayer(mg);       // desenhar antes o polígono!
var collection = ee.ImageCollection('LANDSAT/LC08/C01/T1_TOA')
    .filterBounds(mg)
    .filterDate('2015-04-01', '2018-07-01')
    .filterMetadata('CLOUD_COVER', 'less_than',2);

var median = collection.median()
    .clip(mg);

var imageVisParam =
{"bands":["B4","B3","B2"],"min":0.061,"max":0.11};

Map.addLayer(median,imageVisParam,'landsat 8');
print(median)
```

Exercício 1– Crie uma *layer* de declividade a partir do MDE SRTM 30m. A função para calcular a declividade é ee.Terrain.slope(). Adicione o resultado no mapa. Seu código deve ter no máximo três linhas. Não é necessário definir a área de análise.

Script 2 – Criando um mapa de NDVI com imagens Landsat 8

Os índices de vegetação são amplamente utilizados em estudos ecológicos e ambientais. Eles relacionam os valores numéricos das bandas, normalmente vermelho e infravermelho próximo, resultando num índice que expressa a cobertura foliar fotossinteticamente ativa no pixel. O NDVI é um dos índices mais utilizados. Seus valores variam entre 0 e 1, sendo que valores iguais a zero representam cobertura não vegetal e valores positivos em escala crescente refletem o aumento da área foliar e biomassa vegetal. A fórmula para o cálculo do índice é:

$$NDVI = (NIR – RED) / (NIR + RED)$$

NIR = Reflectância na região espectral do Infravermelho próximo
RED = Reflectância na região espectral do Vermelho

No próximo exemplo iremos calcular o NDVI de três formas diferentes. Algumas funções novas serão utilizadas. Primeiro, vamos fazer uma abstração do nosso código, ordenando as tarefas que ele deve executar para alcançarmos o objetivo. Chamamos isso de **pseudocódigo**.

É desejável que cada linha do pseudocódigo possa ser traduzida em uma função simples da linguagem de programação. Essas informações podem (e devem) ser utilizadas posteriormente na **documentação** do código. Documentar significa descrever **sucintamente** o que cada função do programa faz, através de comentários colocados antes das mesmas. Vejamos um exemplo:

- Criar um ponto no mapa;
- Carregar coleção de imagens Landsat 8 (ID: ‘LANDSAT/LC8_L1T’);
- Filtrar a coleção por data das imagens (‘2015-01-01’ a ‘2016-01-01’);
- Filtrar a coleção por geometria (usar o ponto criado);
- Criar mosaico sem nuvens;
- Selecionar banda R (‘B4’, Red (0.64 - 0.67 µm));
- Selecionar banda NIR (‘B5’, Near Infrared (0.85 - 0.88 µm));
- Calcular NDVI;
- Adicionar imagem NDVI no mapa;

Vamos ao código! Desde o carregamento das imagens até a filtragem, seguiremos os mesmos passos feitos no exercício anterior, com a diferença que agora criaremos uma geometria do tipo ponto para definir o local de análise.

```
var geometry = ee.Geometry.Point([-42.77, -20.78]);

var landsat = ee.ImageCollection('LANDSAT/LC8_L1T')
  .filterBounds(geometry)
  .filterDate('2015-01-01', '2016-01-01')
```

Até aqui, nada de novo! Adicionamos a geometria, carregamos a coleção e filtramos as imagens por data e local. Mas, no próximo passo, vamos utilizar um método diferente para compor o mosaico sem nuvens com imagens Landsat. Veja que dessa vez não aplicamos o filtro por cobertura de nuvens. Ou seja, provavelmente imagens com muitas nuvens estão presentes na coleção.

Existe um algoritmo que foi desenvolvido exatamente para essa função: **ee.Algorithms.Landsat.simpleComposite()**. Esse algoritmo usa as imagens Landsat com o menor processamento (*Raw*), converte-as para reflectância no topo da atmosfera (TOA) e aplica o algoritmo Landsat.simpleCloudScore para obter a mediana dos pixels menos nublados. Somente os pixels com menor cobertura de nuvens são adicionados na imagem final.

Adicionalmente, passaremos para a função o argumento 'asFloat: true', para que o valor das imagens carregadas estejam no formato decimal (*float*). Outros argumentos da função podem ser visualizados ao autocompletar o código (ctrl + espaço). O formato que usaremos para passar os argumentos é: **"parâmetro: valor"**. Eles serão alocados em um dicionário:

{collection: landsat, asFloat: true}

Adicione as seguintes linhas ao código:

```
var LSC = ee.Algorithms.Landsat.simpleComposite({
    collection: landsat,
    asFloat: true
})
Map.addLayer(LSC,'' ,'Landsat')
```

Agora, temos que calcular o NDVI. A primeira forma é criando uma variável para representar cada banda (B4 e B5) e, depois, calculando o índice com funções de álgebra de mapas (subtração, soma, divisão). Para selecionar uma ou mais bandas de uma imagem, usamos o método **select()**. Ele pode ser usado tanto para coleções de imagens quanto para uma única imagem. Vejamos:

```
//Método 1
var b4 = LSC.select('B4') // seleciona banda B4 da imagem
var b5 = LSC.select('B5') // seleciona banda B5 da imagem
```

Agora, basta calcular o índice. Lembre-se que ‘b4’ e ‘b5’ são objetos Earth Engine e requerem métodos adequados para o processamento. Em outras palavras, não podemos simplesmente declarar “var ndvi = (b5 – b4) / (b5 + b4)”, como intuitivamente tenderíamos a fazer. Métodos para cálculos com imagens estão no pacote de algoritmos **ee.Image**. Vamos calcular o NDVI e adicionar o resultado na tela. Configure a visualização da camada para valor mínimo 0 e máximo 1.

```
var ndvi = b5.subtract(b4).divide(b5.add(b4))
```

Realmente, não é uma forma muito simples de escrever uma expressão matemática. Por isso, há uma função que facilita muito essa etapa, permitindo usar a sintaxe comum de operações matemáticas para objetos EE. De qualquer forma, precisamos passar para a função quais são as bandas incluídas no cálculo:

```
//Método 2
var ndvi2 = LSC.expression("(b5 - b4)/ (b5 + b4)",{
  b4: LSC.select("B4"),
  b5: LSC.select("B5")
})
```

Veja que, nesse caso, b4 e b5 são chaves dentro de um dicionário que é passado para o método expression() como um argumento. Poderíamos ter atribuído qualquer nome, uma vez que em seguida declaramos qual banda da imagem é representada por essas variáveis. No exemplo anterior, criamos uma **variável** para cada banda.

Há um método ainda mais simples para calcular o NDVI de uma imagem: **normalizedDifference()**. Esse método é usado para calcular a diferença normalizada entre duas bandas e deve ser aplicado a imagens individuais, como é o caso da camada “LSC”. Os argumentos exigidos pelo método são somente os nomes das bandas NIR e RED da imagem.

```
//Método 3
var ndvi3 = LSC.normalizedDifference(["B5", "B4"])


Map.addLayer(ndvi,'', 'NDVI') //alterar qual ndvi será exibido
```

Pronto! No próximo exercício, veremos como alterar a paleta de **cores** para melhor representação do NDVI.

SCRIPT 2

```
var geometry = ee.Geometry.Point([-42.77, -20.78]);

var landsat = ee.ImageCollection('LANDSAT/LC8_L1T')
.filterBounds(geometry)
.filterDate('2015-01-01', '2016-01-01')

var LSC = ee.Algorithms.Landsat.simpleComposite({
collection: landsat,
asFloat: true
})
Map.addLayer(LSC,'' ,'Landsat')


//Método 1
var b4 = LSC.select(“B4”) // seleciona banda B4 da imagem
var b5 = LSC.select(“B5”) // seleciona banda B5 da imagem
var ndvi1 = b5.subtract(b4).divide(b5.add(b4))

//Método 2
var ndvi2 = LSC.expression("(b5 - b4)/ (b5 + b4)",{
b4: LSC.select("B4"),
b5: LSC.select("B5")
})

//Método 3
var ndvi3 = LSC.normalizedDifference(["B5", "B4"])

// adicionar resultado (modificar a imagem que será adicionada)
Map.addLayer(ndvi1,'', 'NDVI')
```

Script 3 – Série temporal NDVI, exportação de imagens e gráficos

Com o código anterior, o que fizemos foi compor uma imagem que é resultado da combinação de uma série de imagens LS8 ao longo de um ano (2015 - 2016). Isso significa que pixels observados em diferentes épocas do ano estão juntos na mesma imagem. Essa informação pode ser valiosa quando nosso objetivo é realizar uma classificação para separar somente a vegetação das demais classes. Ainda assim, nesse caso seria mais conveniente considerar somente os pixels de valor **máximo** da série. Mas, na prática, geralmente temos interesse em examinar a **variação** desse índice ao longo do tempo.

Agora, veremos como aplicar uma **função** à uma coleção de imagens para calcular o NDVI por cena e incluir o resultado como uma banda **adicional** em cada uma delas. Depois, criaremos um **gráfico** que irá representar a variação do índice ao longo do **tempo**.

Iremos criar uma geometria e adicionar a coleção Landsat 8, da mesma forma que fizemos anteriormente, mas dessa vez aplicando o filtro de nuvens através dos **metadados** e para um período mario. Para a série Landsat 8, o campo dos metadados que guarda a informação sobre cobertura de nuvens numa cena é 'CLOUD_COVER'. Selecionaremos apenas imagens com menos de 20% de cobertura de nuvens.

```
var geometry = ee.Geometry.Point([-43.069, -20.488]);
var landsat = ee.ImageCollection('LANDSAT/LC08/C01/T1')
          .filterBounds(geometry)
          .filterDate('2013-08-01', '2018-01-01')
          .filterMetadata('CLOUD_COVER', 'less_than',20)
```

Em seguida, criaremos uma **função** que irá tomar uma imagem, calcular o NDVI e adicionar o resultado como uma banda adicional dessa mesma imagem. Usaremos novamente o método **normalizedDifference()**, que vimos no exercício anterior, para calcular o índice. Usaremos também o método **rename()** para alterar o nome da imagem resultante. Para adicionar o resultado (camada ndvi) como uma banda na imagem original, usaremos o método **addBands().**

```
var calcNDVI = function(imagem) {
    var result = imagem.normalizedDifference(["B5",
"B4"]).rename('ndvi');
    return imagem.addBands(result)
}
```

Agora, o que precisamos fazer é aplicar essa função para todas as imagens da coleção. Normalmente, a solução mais comum em programação é usar um iterador *for* para percorrer cada elemento de uma lista (no caso, cada imagem da coleção) e aplicar a função. No entanto, essa lógica não é a utilizada no Earth Engine. No lugar de iteradores, é empregado o método **map()**, que funciona exatamente da mesma forma. Quando utilizamos esse método, ele percorre cada cena da coleção e aplica a função desejada. A sintaxe é:

coleção**.map**(função)

No nosso exemplo, a coleção é 'landsat' e a função é 'calcNDVI'. Vamos aplicar a função e, em seguida adicionar o resultado ao mapa.

```
var ndvi = landsat.map(calcNDVI)
Map.addLayer(ndvi.select('ndvi'), {min: 0, max:1})
```

O que temos é uma nova coleção de imagens denominada 'ndvi'. Essa coleção foi gerada a partir da coleção original ao aplicar a função calcNDVI, que nós declaramos no código. O nome da banda que contém o índice calculado também é 'ndvi'. Por isso, ao adicionar a camada ao mapa, usamos o método **select()** para selecionar somente essa banda.

Abra o inspetor, clique em algum ponto do mapa e veja as informações. Clique em *Series* e veja o gráfico com a variação do valor NDVI ao longo do tempo para o ponto onde você clicou.

Observe que não precisamos passar nenhum argumento para a função calcNDVI, porque nesse caso, está sendo empregada como um método aplicado à imagem 'landsat'.

Aplicando paleta de cores

Vamos aplicar uma paleta de cores para melhorar a representação do NDVI. No navegador, clique com o botão direito na aba onde está aberto o Earth Engine e duplique a aba. Na nova aba, em *Scripts*, procure por 'NDVI'. Em *examples -> image*, abra o script "Normalized Difference". Copie da linha 14 a 17 e cole no seu código:

```
// Make a palette: a list of hex strings.
var palette = ['FFFFFF', 'CE7E45', 'DF923D', 'F1B555', 'FCD163',
'99B718','74A901', '66A000', '529400', '3E8601', '207401', '056201',
                '004C00', '023B01', '012E01', '011D01', '011301'];
```

Altere a linha de comando que adiciona a imagem NDVI ao mapa, indicando paleta de cores a ser aplicada. Para melhor representação, usaremos o valor **mediano** dos pixels de toda a coleção.

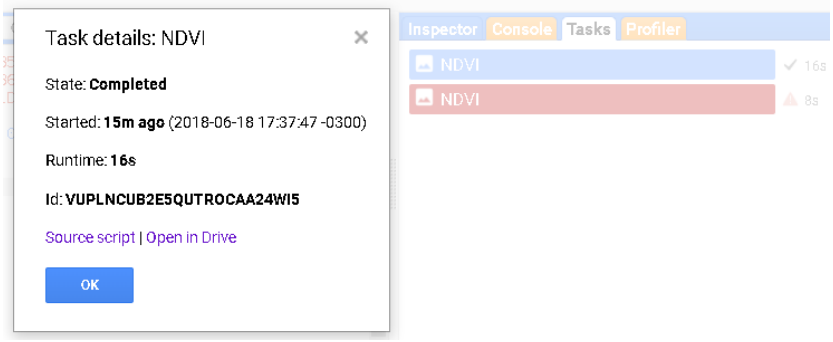
```
Map.addLayer(ndvi.select('ndvi').median(), {min: 0, max:1, palette: palette})
```

Agora, iremos **exportar** a imagem para possibilitar seu download pelo usuário. O Earth Engine oferece três possibilidades para a exportação, veja mais detalhes na aba *Script*¸ buscando pelo termo '*export*'.

Iremos exportar a imagem para o Google Drive. Antes, definiremos uma área a partir de um ***buffer*** de 5km do ponto registrado em 'geometry' e, em seguida, gerar um quadrado a partir do círculo criado. Assim, exportaremos somente uma parte da imagem, para acelerar o processamento. Utilizaremos uma nova função, **Map.centerObject()**, para centralizar o mapa nesse quadrado.

```
var regiao = geometry.buffer(5000).bounds()
Map.addLayer(regiao)
Map.centerObject(regiao)

Export.image.toDrive({
    image: ndvi.select('ndvi').median(),
    description: "NDVI",
    region: regiao,
    scale: 30,
    crs: "EPSG:4326"

})
```

Quando o processamento terminar, a imagem exportada será adicionada na aba "*Tasks*". Clique em "RUN" para visualizar as configurações de exportação. Deixe as configurações padrão e clique novamente em "Run". Esse procedimento pode demorar bastante dependendo do tamanho das imagens. Ao finalizar, clique ícone botão de interrogação para obter o link do arquivo exportado e ver informações do processamento. Faça o *download* da imagem e veja o resultado.



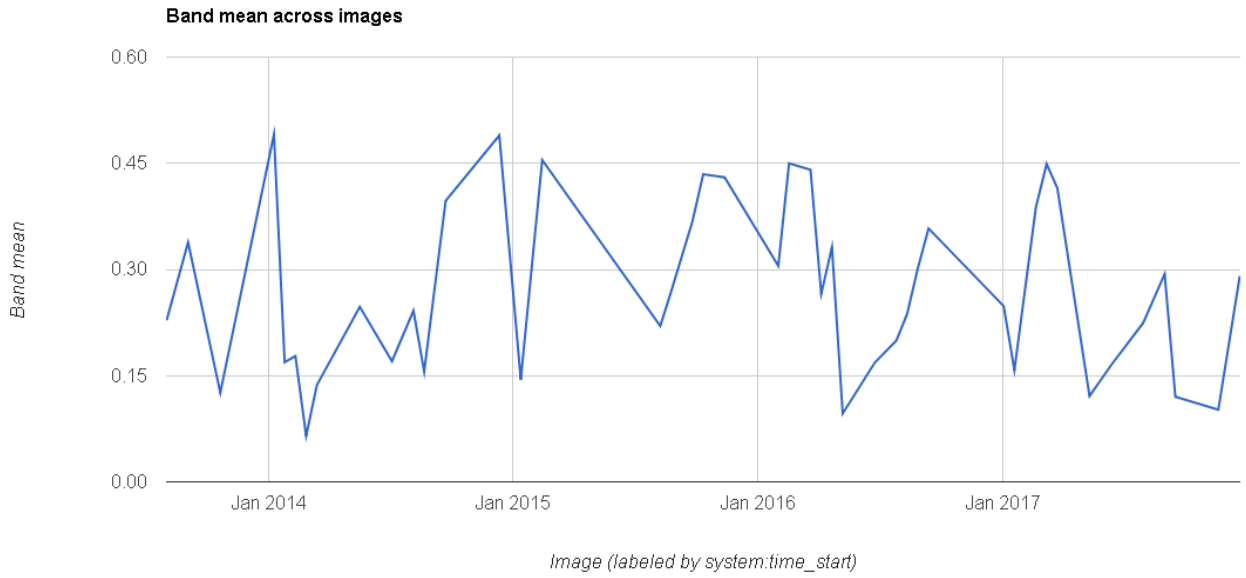
Enquanto aguardamos a exportação da imagem, vejamos alguns tipos de gráficos para séries temporais e como exportá-los.

Gráfico tipo 1 – Série temporal completa

O tipo mais elementar de gráfico exibe a série temporal completa, com todas observações ordenadas no tempo, para um local definido. Esse gráfico é gerado com a função ui.Chart.image.series. Para imprimir o gráfico usamos a função print.

```
print(ui.Chart.image.series(ndvi.select('ndvi'),geometry,ee.Reducer.mean()))
```

O gráfico a seguir será adicionado ao console:



No ponto que estamos analisando, o NDVI máximo ocorre em janeiro e o mínimo em agosto. Entretanto, contrariando o esperado, no ano de 2015 o NDVI foi muito baixo no mês de janeiro. Isso ocorre devido à presença de **nuvem** no pixel na data da observação. Mais adiante, veremos como usar uma **máscara** de nuvens e sombras para não incluir pixels com nuvens na série temporal.

Você pode baixar os dados dessa série facilmente em uma tabela CSV, caso tenha interesse em processar os dados no R, por exemplo. Basta clicar no ícone de exportação no console (acima, à direita) e clicar no botão 'Download CSV'.

Observação: Caso o usuário não determine a região (ponto ou polígono) de análise, o EE irá considerar toda a imagem. Porém o limite de processamento é de 1 milhão de pontos.

Gráfico tipo 2 – Série compilada por dia do ano

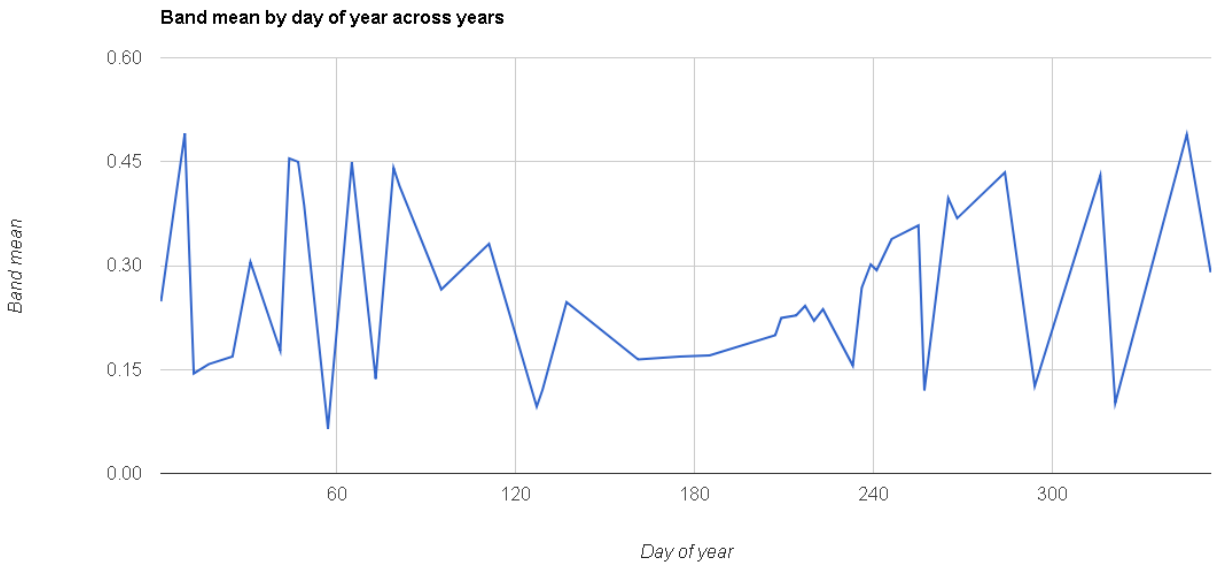
Para exportar um gráfico que compila a série temporal num único ano, usaremos a função ui.Chart.image.doySeries. Vamos **exportar** um gráfico com essa série temporal para o ponto definido em 'geometry'. A função **ui.Chart.image.doySeries()** é utilizada para criar esse tipo de gráfico.

Adicione essa linha ao código para imprimir o gráfico:

```
print(ui.Chart.image.doySeries(ndvi.select('ndvi'),geometry,ee.Reducer.mean(),1))
```

No exemplo, para a banda 'ndvi' da série ndvi, ela toma o valor **médio** da variável (*vide* documentação), para uma região definida. Portanto, o que vemos no gráfico é o valor médio do NDVI por dia do ano, para o ponto definido em *geometry,*.

Como há, possivelmente, registro do mesmo dia do ano em diferentes anos, os valores são reduzidos para um valor único usando a média (**ee.Reducer.mean()**). Como se trata de um ponto, a média é o valor observado no próprio ponto. Caso utilizássemos um polígono, o valor resultante seria a média dos pixels dentro deste, por dia. No console, você verá este gráfico:



Portanto, esse gráfico mostra o padrão anual de variação do NDVI no ponto que determinamos.

Gráfico tipo 3 – Séries temporais por ano

Nesse tipo de gráfico, cada ano presente na série será plotado no gráfico como uma linha diferente. Seu grande potencial é para a comparação entre anos e a detecção

de anos atípicos, por exemplo. Usaremos a função ui.Chart.image.doySeriesByYear. Para simplificar, deixaremos os parâmetros opcionais como *default.*

```
print(ui.Chart.image.doySeriesByYear(ndvi, 'ndvi', geometry))
```

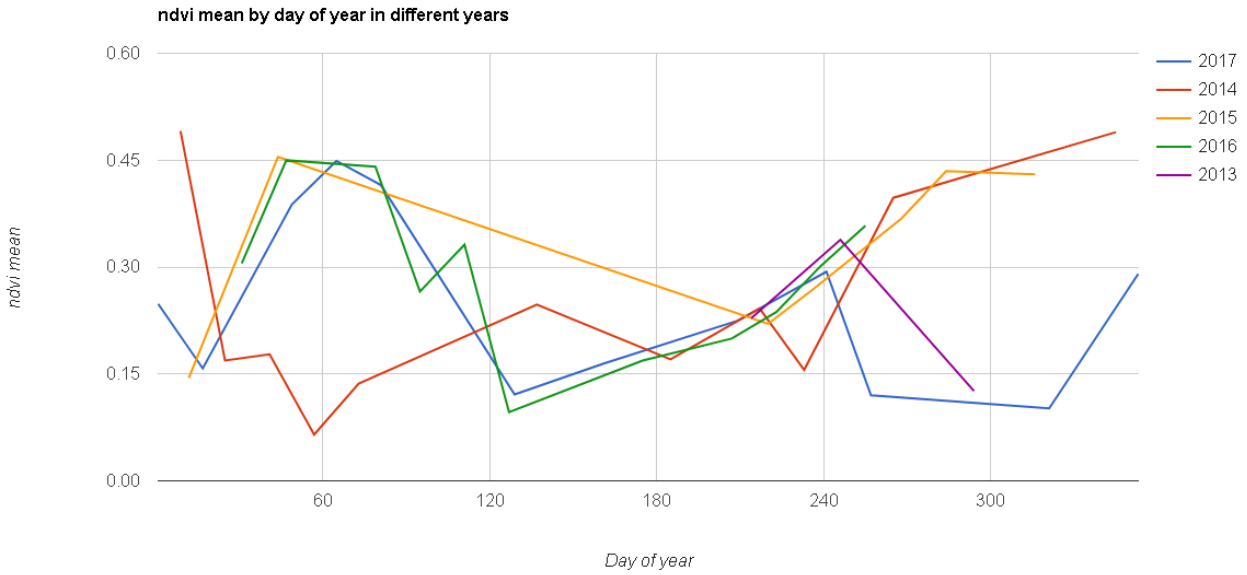


Gráfico tipo 4 – Séries temporais de diferentes locais

O último tipo de gráfico que abordaremos será criado para diferentes pontos de observação, definidos pelo usuário. Com esse tipo de informação, podemos comparar a variação de diferentes tipos de cobertura do solo ao longo do tempo.

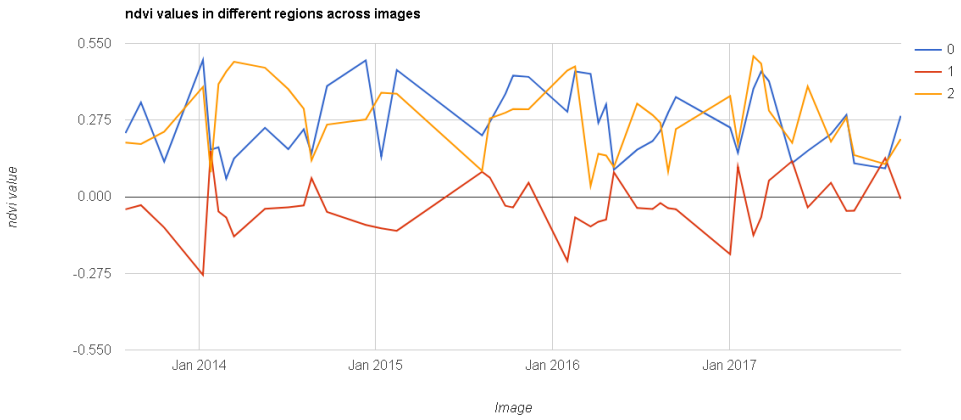
Primeiro, temos que criar geometrias do tipo ponto em mais dois locais com cobertura de solo distintas (o ponto já adicionado no código está sobre uma mata). Adicione um ponto sobre um corpo d'água e outro sobre uma pastagem.

```
var geometry = ee.Geometry.MultiPoint([[-43.069603443145695, -20.48897451548106],
        [-43.28355791280046, -20.47602951589711],
        [-43.291218286030926, -20.538182021114466]]);
```

Depois converta a geometria para o tipo "Feature Collection", a qual conterá os três pontos.

Usaremos a função ui.Chart.image.seriesByRegion para criar o gráfico:

```
print(ui.Chart.image.seriesByRegion(ndvi,geometry,ee.Reducer.mean(), 'ndvi'))
```

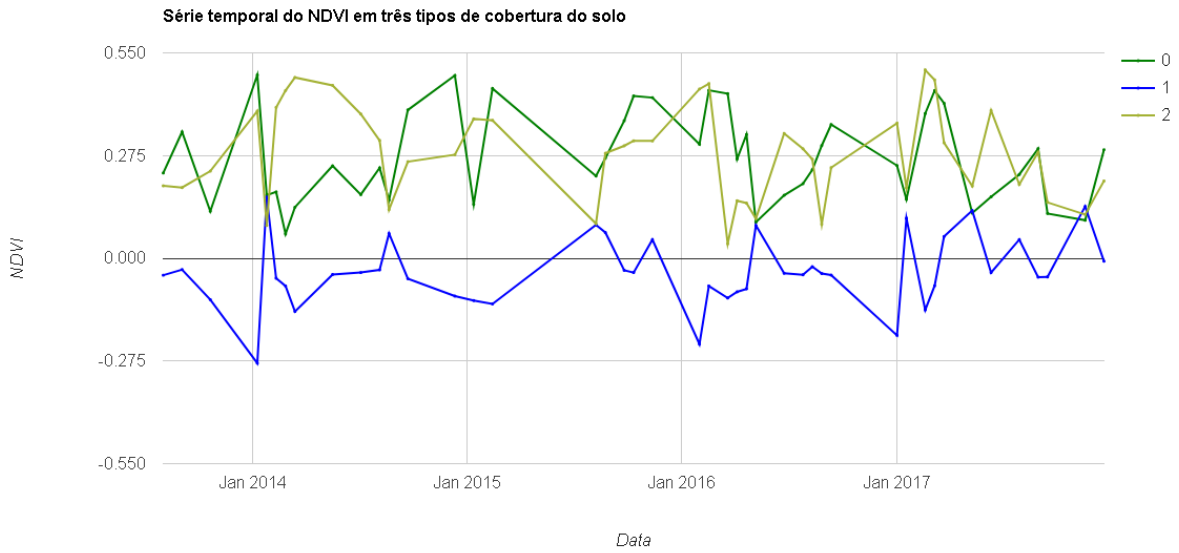


Temos, portanto, a variação do NDVI de cada ponto de observação representado numa linha. Mas, algumas coisas poderiam ser melhoradas nesse gráfico. Por exemplo, poderíamos determinar o nome de cada tipo de cobertura e ilustrar cada uma com uma cor mais intuitiva para melhorar a interpretação (e.g mata – verde, água – azul). Poderíamos também alterar o título do gráfico, o nome de cada eixo, os valores de cada eixo, espessura das linhas, etc. Todas essas configurações são feitas com o método setOptions() aplicado ao objeto gráfico. Primeiro, criaremos uma variável que conterá todas as configurações do gráfico e posteriormente passaremos essas configurações para o método setOptions:

```
var options = {
  title: 'Série temporal do NDVI em três tipos de cobertura do solo',
  hAxis: {title: 'Data'},
  vAxis: {title: 'NDVI'},
  lineWidth: 2,
  pointSize: 1,
  series: {
    0: {color: '008000'}, // mata
    1: {color: '0000FF'}, // agua
    2: {color: 'A4B235'}, // pastagem
}};
```

Vamos imprimir novamente o gráfico com as opções que configuramos:

```
print(ui.Chart.image.seriesByRegion(ndvi,geometry,ee.Reducer.mean(
),'ndvi')
  .setOptions(options)
  )
```



Veja que o rótulo ainda não exibe as propriedades do ponto (tipo de cobertura). Isso aconteceu pois em nenhum momento declaramos qual é o tipo de cobertura de cada ponto. Ou seja, os pontos não possuem nenhum **atributo** além da sua posição geográfica. No próximo script, veremos como alterar as propriedades de uma coleção de feições. Nesse caso, o rótulo das séries apareceria automaticamente no gráfico.

Há uma infinidade de tipos de gráficos e opções no Earth Engine. Para mais informações sobre a customização de gráficos, acesse: https://developers.google.com/chart/interactive/docs/

Script 3

```
var geometry = ee.Geometry.Point([-42.77, -20.78]);

var landsat = ee.ImageCollection('LANDSAT/LC08/C01/T1')
.filterBounds(geometry)
.filterDate('2013-08-01', '2018-01-01')
.filterMetadata('CLOUD_COVER', 'less_than',50);

var calcNDVI = function(imagem) {
    var      result      =      imagem.normalizedDifference(["B5",
"B4"]).rename('ndvi');
    return imagem.addBands(result)
}

var ndvi = landsat.map(calcNDVI)

// Paleta de cores:
var palette = ['FFFFFF', 'CE7E45', 'DF923D', 'F1B555', 'FCD163',
'99B718','74A901', '66A000', '529400', '3E8601', '207401', '056201',
               '004C00', '023B01', '012E01', '011D01', '011301'];

Map.addLayer(ndvi.select('ndvi').median(), {min: 0, max:1, palette:
palette},'NDVI')

/*var regiao = geometry.buffer(5000).bounds()
Map.addLayer(regiao,'','regiao')
Map.centerObject(regiao)

Export.image.toDrive({
    image: ndvi.select('ndvi').median(),
    description: "NDVI",
    region: regiao,
    scale: 30,
    crs: "EPSG:4326"
})
*/

//Gráfico tipo 1 - Série completa
print(ui.Chart.image.series(ndvi.select('ndvi'),          geometry,
ee.Reducer.mean()))

//Gráfico tipo 2 - Série compilada por dia
print(ui.Chart.image.doySeries(ndvi.select('ndvi'),geometry,ee.Red
ucer.mean(),1))
```

```
//Gráfico tipo 3 - Séries por ano
print(ui.Chart.image.doySeriesByYear(ndvi, 'ndvi', geometry))

//Gráfico tipo 4 - Séries para diferentes localizações
// Adicionar pontos e Converter para Feature Collection manualmente
var geometry = ee.Geometry.MultiPoint(
        [[-43.069603443145695, -20.48897451548106],
         [-43.28355791280046, -20.47602951589711],
         [-43.291218286030926, -20.538182021114466]]);

// Configurações do gráfico 4
var options = {
  title: 'Série temporal do NDVI em três tipos de cobertura do solo',
  hAxis: {title: 'Data'},
  vAxis: {title: 'NDVI'},
  lineWidth: 2,
  pointSize: 1,
  series: {
    0: {color: '008000'}, // mata
    1: {color: '0000FF'}, // agua
    2: {color: 'A4B235'}, // pastagem
}};

//Imprimir gráfico 4
print(ui.Chart.image.seriesByRegion(ndvi,geometry,ee.Reducer.mean(
),'ndvi')
  .setOptions(options)
  )
```

Script 4 – Classificação supervisionada de imagens

A plataforma Earth Engine conta com uma série de algoritmos classificadores de imagens. Os algoritmos mais utilizados para classificação estão disponíveis para uso no seu código. Na aba 'Docs', busque o termo '*classifier*' e explore as possibilidades.

Os algoritmos podem ser usados para três tipos de predição:

- Classificação, quando retorna valores de variáveis discretas (classes);
- Regressão, quando retorna valores contínuos;
- Probabilidade, quando retorna a probabilidade do classificador estar correto.

Uma possibilidade dos classificadores no EE é treiná-los com diferentes conjuntos de amostras, de forma acumulativa. Assim, os classificadores podem ser atualizados com novas amostras coletadas, sem alterar as anteriores. Essa alternativa também é importante quando trabalhamos com volumes muito grandes de dados, uma vez que a plataforma possui **limite** de processamento (1 milhão de pontos). Nesses casos, podemos dividir nossas amostras de treinamento em frações menores e evitar problemas de processamento.

Mas, vamos começar com o básico! Iremos fazer uma classificação simples, para entendermos o funcionamento da plataforma para esse tipo de tarefa.

Primeiro, vamos adicionar um polígono para delimitar nossa área de análise e, em seguida, adicionar a coleção Sentinel "COPERNICUS/S2". Filtraremos essa coleção para nossa área de análise e selecionaremos imagens com o mínimo de cobertura de nuvens.

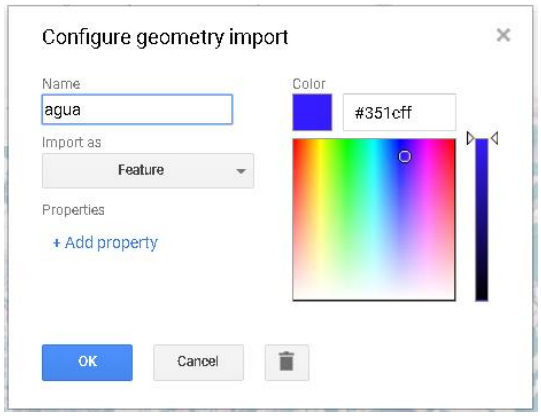
```
var aoi = ee.Geometry.Polygon(
        [[[-48.495025634765625, -8.594611757619093],
          [-47.772674560546875, -8.602758887410024],
          [-47.775421142578125, -8.146254441291376],
          [-48.486785888671875, -8.147613872924316]]])

var sentinel = ee.ImageCollection("COPERNICUS/S2")
  .filterBounds(aoi)
  .filterMetadata('CLOUD_COVERAGE_ASSESSMENT', 'less_than', 1)

Map.addLayer(sentinel,'','Sentinel')
```

Veja que a coleção possui 17 imagens (use o inspetor). No entanto, queremos usar somente uma imagem de alta qualidade para realizar a classificação. Usaremos então um **redutor** para obter uma única imagem com o valor mediano dos pixels e depois a cortaremos para nossa área de estudo.

```
var img = sentinel.median().clip(aoi)
```

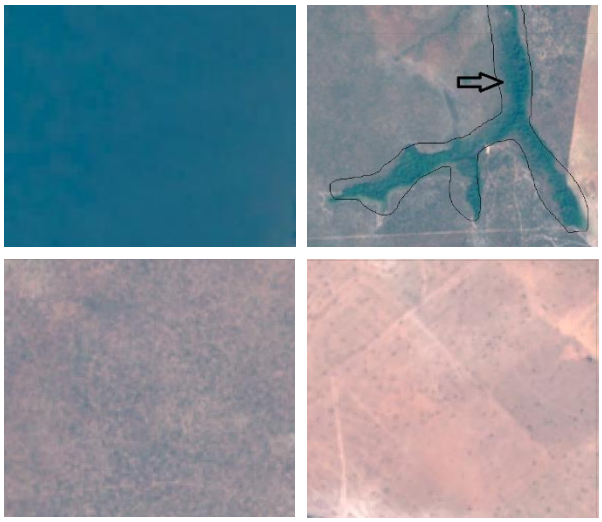
O próximo passo é a coleta de amostras para treinamento do classificador. Podemos fazer isso de diferentes formas. No nosso primeiro contato com o classificador, iremos usar a própria plataforma do EE para coletar as amostras.

Crie uma nova geometria do tipo ponto (Adicionar Marcador ou Geometry Imports -> New Layer). **Cada tipo** de cobertura a ser classificada terá suas amostras guardadas em uma **geometria nova**. Nomeie a geometria como 'agua' e configure a importação como "Feature". Você pode alterar a cor dos pontos para melhor visualização, no caso da água mude a cor para azul. Clique em OK.



Agora, basta clicar na imagem para coletar as amostras. Na área de estudo, colete amostras para as classes: água, mata, campo cerrado e solo exposto. Os padrões a serem considerados são ilustrados a seguir. Percorra a imagem e selecione **20** amostras de cada classe.

Dica: para excluir um ponto adicionado, pressione "Esc" para sair do modo de edição de pontos, clique sobre o ponto e pressione "Del"



Ao terminar a coleta de amostras, você terá que criar um **identificador** para cada geometria, associando um valor único para cada classe de cobertura. Na seção de importação, clique no botão "Configure". Na janela de configuração, clique em "+ Add Property". No campo 'name', escreva 'class', e em 'value', associe um valor numérico para cada classe: 1 – agua, 2 – mata, 3 – campo-cerrado e, 4 – solo.

Em seguida, iremos juntar essas feições, ou geometrias, (features) em uma única classe de feições. Usaremos o comando **merge()** para uni-las, mas antes precisamos de uma classe de feições já existente para isso. Então, iremos converter alguma das feições criadas para o formato FeatureCollection e, depois, anexar as outras feições à ela (a ordem não importa).

```
var amostras = ee.FeatureCollection(agua)
  .merge(mata)
  .merge(campo_cerrado)
  .merge(solo_exposto)
```

A nova variável 'amostras' é uma coleção de feições que contém os pontos coletados. O próximo passo será definir quais bandas serão usadas na classificação. Caso queira usar todas as bandas do sensor, essa etapa não é necessária. Todavia, na prática, geralmente deixamos algumas bandas de fora da classificação.

Neste exemplo, usaremos as bandas das regiões espectrais do visível (RGB) e infravermelho próximo. Na coleção que estamos utilizando, correspondem às bandas 4, 3, 2 e 8, respectivamente. Criaremos uma nova variável do tipo lista para declarar as bandas que utilizaremos:

```
var bandas = ['B4', 'B3', 'B2', 'B8']
```

Agora, o que faremos é **treinar** o classificador. Primeiro, criaremos uma variável que irá conter as amostras (pontos amostrados) juntamente ao **valor numérico** das bandas selecionadas, para cada ponto. Usaremos a função **sampleRegions()** para associar esse valores aos pontos:

```
var treino = img.select(bandas).sampleRegions({
  collection: amostras,
  properties: ['class'],
  scale: 20             //scale é a resolução da imagem, em metros!
})
```

O **treinamento** do modelo será feito criando uma nova variável que irá armazenar o classificador já treinado com as amostras coletadas. Aqui, usaremos o classificador Random Forests (**ee.Classifier.randomForests()**) e a função **train()** para treinamento.

Iremos passar somente os argumentos básicos para a função train(). Outras opções serão deixadas na configuração padrão. Mas, é recomendável estudar a função e conhecer as opções de processamento.

```
var rf = ee.Classifier.randomForest().train({
  features: treino,          // amostras
  classProperty: 'class',    // campo da tabela que contém a classe
  inputProperties: bandas    // bandas que serão utilizadas
})
```

Por fim, basta aplicar o classificador que acabamos de treinar na nossa imagem. A classificação é feita com o método **classify()**, cujos argumentos são o classificador (rf , no nosso caso). Adicionalmente, podemos alterar o nome da propriedade que irá conter o valor da classificação. Vamos rodar o classificador!

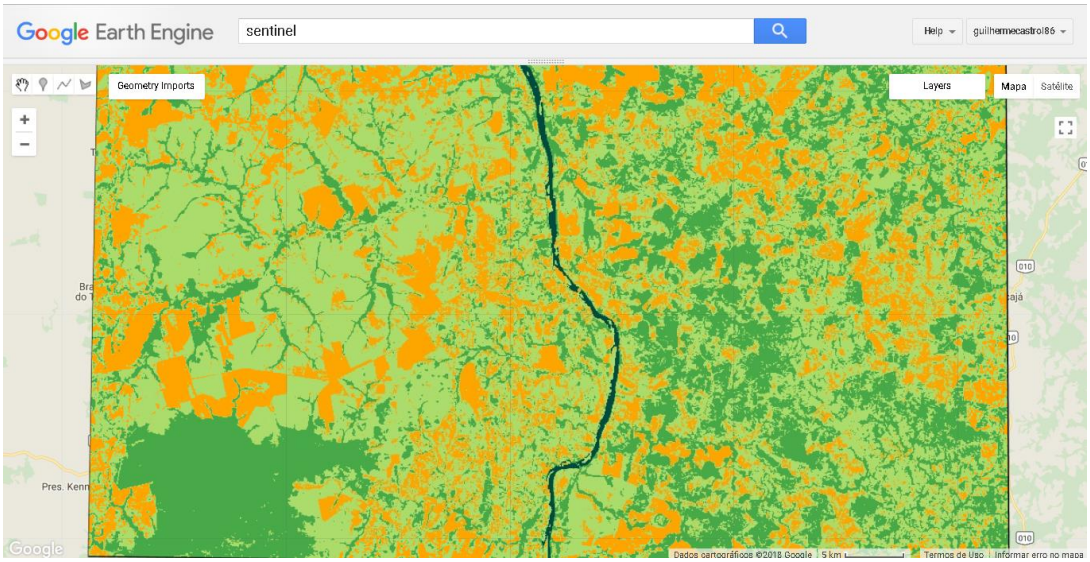
```
var classificacao = img.select(bandas).classify(rf, 'classe')
```

Agora, iremos adicionar a imagem resultante da classificação ao mapa, configurando os parâmetros de visualização para os valores mínimo e máximo das classes (1-4) e determinando as cores de cada classe no mapa.

```
Map.addLayer(classificacao, {min: 0, max:4, palette: ['blue', 'darkgreen', 'lightgreen', 'orange']}, 'Classificacao')

Map.addLayer(img, {bands:['B4','B3','B2']},'Sentinel')
```

Dica: O Earth Engine trabalha com dois padrões para o sistema de cores. Podemos declarar o nome da cor (como fizemos agora) ou usar códigos hexadecimais para o padrão de cores (como fizemos no script 3). Acesse http://www.color-hex.com/ para ver qual é o código correspondente da cor que você deseja utilizar.

Eis a nossa classificação:



Exercício 2: Faça uma nova classificação usando o classificador ee.Classifier.cart. Exporte o novo mapa para uma nova variável. Adicione as duas classificações ao mapa e compare. Responda às seguintes questões:

a) Caso seu objetivo seja mapear trechos seguramente navegáveis do rio Tocantins, dentro da área de análise, qual dos dois classificadores lhe fornece um mapa mais confiável?

b) Caso seu objetivo seja mapear a área total das matas ripárias na região, qual classificador apresentou melhores resultados?

Script 4

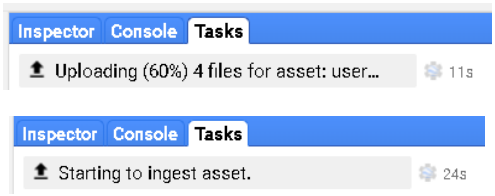
```
var aoi = ee.Geometry.Polygon(
        [[[-48.495025634765625, -8.594611757619093],
          [-47.772674560546875, -8.602758887410024],
          [-47.775421142578125, -8.146254441291376],
          [-48.486785888671875, -8.147613872924316]]]),

Map.addLayer(aoi,'','AOI');

var sentinel = ee.ImageCollection("COPERNICUS/S2")
  .filterBounds(aoi)
  .filterMetadata('CLOUD_COVERAGE_ASSESSMENT', 'less_than', 1);

var img = sentinel.median().clip(aoi);

// Coletar amostras Antes! Criar uma Feature separada para cada tipo.

var amostras = ee.FeatureCollection(agua)
  .merge(mata)
  .merge(campo_cerrado)
  .merge(solo_exposto);

var bandas = ['B4', 'B3', 'B2', 'B8'];

var treino = img.select(bandas).sampleRegions({
  collection: amostras,
  properties: ['class'],
  scale: 20
});

var rf = ee.Classifier.randomForest().train({
  features: treino,
  classProperty: 'class',
  inputProperties: bandas
})

var classificacaorf = img.select(bandas).classify(rf)

Map.addLayer(classificacaorf, {min: 0, max:4, palette: ['blue',
'darkgreen', 'lightgreen', 'orange']}, 'Classificacao')
Map.addLayer(img, {bands:['B4','B3','B2']},'Sentinel')
```

Importação de dados

Até o momento, usamos somente dados e geometrias disponíveis no EE, ou criadas manualmente no próprio sistema. Mas, muitas vezes, queremos utilizar arquivos que já temos em mãos para realizar os processamentos, como limites, hidrografias, ou pontos de amostragem e validação de uma classificação. Veremos como enviar um arquivo vetorial (*shapefile)* para o sistema e, em seguida, importa-lo para o código. Há duas formas diferentes de fazer o upload de arquivos.

Carregando diretamente no Earth Engine

Clique na aba 'Assets' e então em 'New' e 'Table Upload'. Clique no botão 'Select' e navegue até o diretório onde está o arquivo shapefile. Selecione as extensões 'shp', 'dbf', 'shx' e 'prj' e abra. Em seguida, altere o nome da pasta em que os arquivos serão salvos. Clique em ok. Observe que na aba 'Tasks', será exibido o processo de upload e, posteriormente, de ingestão do arquivo. Esse procedimento leva alguns minutos.



Observação: Caso você não carregue o arquivo '.prj' que compõe o shapefile, o EE assumirá que o mesmo está georreferenciado no DATUM WGS 84, lat/long.

Uma vez terminada a ingestão, o arquivo estará disponível na sua pasta. Você irá importa-lo declarando o caminho do diretório em que ele foi salvo. O construtor a ser utilizado dependerá se estamos carregando uma coleção de feições (shapefile, por exemplo), uma imagem raster ou uma coleção de imagens. No caso do shapefile, usaremos o construtor **ee.FeatureCollection**. Por exemplo:

```
var caatinga = ee.FeatureCollection('users/usuario/Caatinga_Limite_IBGE')
```

Carregando por Fusion Tables

A outra forma de carregar dados para o EE é através de Fusion Tables.

shapescape.com

Script 5 - Máscara de nuvens

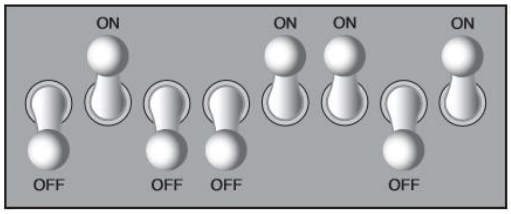
Nos exemplos anteriores, utilizamos um filtro para selecionar cenas com baixa ocorrência de nuvens. O problema dessa abordagem é que, ao desprezar uma cena com cobertura de nuvens acima do limite, muitos pixels em boas condições são excluídos da análise, quando poderiam ser computados normalmente.

Para contornar essa situação, normalmente é utilizada uma banda dos sensores que contém a informação da qualidade do pixel. Além de nuvens, também são informadas as presenças de sombras, neve/gelo, água, entre outros, a depender do sistema.

Antes de aplicar uma máscara, precisamos entender como funciona o padrão dos valores presentes nessas bandas. Em geral, os dados são armazenados em um sistema binário de 8 ou 16 bits. Vejamos rapidamente como isso funciona.

Sistema binário

Imagine que cada bit é uma 'chave' que pode estar ligada (1) ou desligada (0). Na imagem abaixo, temos um sistema de 8 bits. Um conjunto de 8 bits forma um byte.

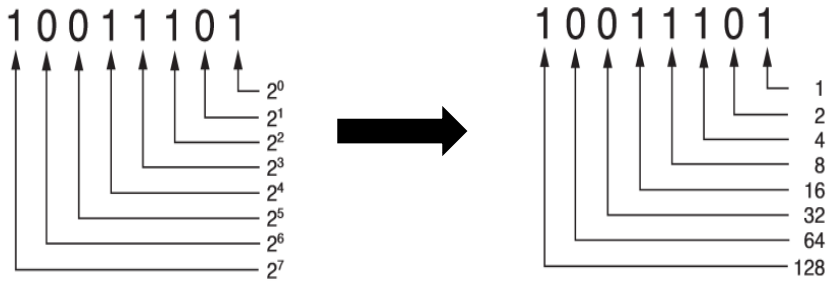


Fonte: Gaddis, 2012.

A posição de cada bit tem um valor específico, que é resultado da conjugação da posição do bit e do seu valor (0 ou 1). Vejamos o seguinte exemplo de um número escrito no sistema binário de 8 bits:

1001 1101

A posição de cada dígito num número binário tem um valor associado à ele em potências de 2. Começando da direta para a esquerda, os valores são: 2^0, 2^1, 2^2, 2^3, até 2^{n-1}, sendo 'n' o número de bits.



Para saber o valor de um número binário, tudo que temos a fazer é **somar** os valores dos bits que estão "**ligados**", ou com valor 1. Por exemplo, no número binário "1001 1101", a posição dos bits 'ligados' equivale aos valores: 1, 4, 8, 16 e 128 (da direita para a esquerda). A soma desses valores é 157, portanto o valor decimal do binário '1001 1101' é 157.

Quando todos os bits estão 'desligados', com valor 0, o valor de um byte é 0. Quando todos estão com valor '1', o valor é o máximo possível armazenado por um byte. No caso de 8 bits é: 1 + 2 + 4 + 8 + 16 + 32 + 64 + 128 = 255. Por esta razão, o valor máximo armazenado em um byte é 255. O total de valores únicos é 256, pois o valor zero também conta. Com 2 bytes (16 bits) teríamos 2^{16} = 65.536 valores possíveis. Para números maiores, serão necessários mais bytes.

Voltemos agora às nossas imagens de satélite. Vamos tomar como exemplo a coleção Landsat 8 Collection 1 Tier 1 TOA Reflectance. Essa coleção possui uma banda denominada 'BQA'. A descrição dos valores contidos nessa banda é feita no Landsat QA Band Support. Vejamos a tabela da página 29 do referido material:

Landsat 8 OLI, OLI/TIRS Collection 1 QA band bits: Read RIGHT to LEFT, starting with Bit 0															
BIT	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DESCRIPTION	Reserved			Cirrus Confidence		Snow/Ice Confidence		Cloud Shadow Confidence		Cloud Confidence		Cloud	Radiometric Saturation		Terrain Occlusion	Designated Fill

Os bits de posição 0, 1 e 4 tem uma única 'chave' (valor 0 ou 1), enquanto os bits de 2, 3 e do 5 ao 12 são duplos. No primeiro caso (bit simples), o valor 1 indica 'sim, essa condição existe' e o valor 0 indica 'não, essa condição não existe'. Para os bits duplos, há diferentes níveis de confiança para aquela condição. Os quatro valores possíveis (00, 01, 10, 11) indicam:

00: 'não determinado' – o algoritmo não processou esse pixel

10: 'não' – há pouca ou nenhuma chance dessa condição existir (0 – 33%)

01: 'talvez' – há uma chance média dessa condição existir (34-66%)

11: 'sim' – há grande chance dessa condição existir (66-100%)

Nessa coleção em específico, os bits 13, 14 e 15 estão reservados para uso futuro.

Por exemplo, o valor de um pixel qualquer na banda 'BQA' dessa coleção é 7104. Traduzindo para o sistema binário, temos: 0001 1011 1100 0000 (você pode fazer essa conversão na calculadora do Windows, no modo Programador). A leitura é feita da direita (bit 0) para a esquerda (bit 15) Associando às condições respectivas de cada bit, temos:

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Condição	Reserved			Cirrus Conf.		Snow		Shadow		Cloud Conf.		Cloud	Rad. Sat.		T.O	D.F
Valor	**0**	**0**	**0**	**1**	**1**	**0**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**

Portanto, o valor numérico 7104 indica que o pixel não está coberto por nuvem, que está sombreado, que não há neve ou gelo e que há cirros.

Se o que nos interessa é remover pixels que estejam com nuvens ou sombreados por elas, no exemplo, esses valores seriam:

- Bit 4 (Cloud): valor 1
- Bit 5 (Cloud Conf): valor 1
- Bit 6 (Cloud Conf): valor 0 **ou** 1 (talvez ou sim)
- Bit 7 (Shadow Confidence): valor 1
- Bit 8 (Shadow Confidence): valor 0 **ou** 1 (talvez ou sim)
- Bit 11 (Cirrus Confidence): valor 1
- Bit 12 (Cirrus Confidence): valores 0 **ou** 1 (talvez ou sim)

Como os valores e as posições dos bits são fixos, eles se repetem cena após cena para as mesmas condições. Assim, podemos usar diretamente valores já conhecidos para condições comuns de pixels. Veja a tabela Apêndice B do Landsat QA Band Support. Podemos ver que há **muitos** valores que representam diversas combinações de condições indesejáveis e **poucos** valores que representam condições favoráveis, ou um pixel 'limpo'. A saber, para essa coleção de imagens, os valores que representam pixels de alta qualidade são: 2720, 2724 e 2728 (*Nota: esses valores podem ser diferentes dependendo da aplicação. Para o nosso exemplo, estamos removendo pixels com nuvens, cirros ou sombreados*).

Por isso, é mais simples selecionar os pixels de boa qualidade do que remover valores que representam pixels de má qualidade. O que precisamos fazer então é criar uma máscara que terá valor 1 onde os pixels são validos e 0 onde serão descartados.

Em seguida, temos que aplicar essa máscara à cada imagem da coleção para remover os pixels inválidos e retornar uma nova coleção de imagens somente com pixels válidos.

Criaremos, portanto, uma função que executará as seguintes tarefas:

- Selecionar a banda que contém dados de qualidade do pixel ('BQA' para LS8 TOA Collection 1)
- Criar um *raster* de valor único, para cada valor de pixel aproveitável
    - Cada valor será alocado em uma banda diferente desse novo *raster*
- Comparar a banda 'BQA' com esses *rasters*. Retorna valor 1 onde valores são iguais e 0 onde são diferentes.
    - Usar um redutor para retornar somente o valor máximo (1) desse teste lógico (essa é a nossa máscara)
- Aplicar máscara e retornar imagem final sem nuvens.

Essa função terá três argumentos:

- Uma imagem, (objeto EE.Image)
- Uma lista de valores de bons pixels na banda BQA (EE.List)
- O nome da banda que representa a qualidade dos pixels (EE.String)

Vejamos passo a passo a construção dessa função. Seu nome será 'uniMask' e tomará como argumentos: image (imagem), maskList (lista) e qaband (string)

Iremos declará-la da seguinte forma:

function uniMask(image, maskList, qaband){}

A primeira tarefa é selecionar, dessa imagem, a banda QA:

var qa = image.select(qaband)

Em seguida, criar um *raster* com valor constante em todos os pixels. Esses valores serão iguais ao que iremos declarar para os bons pixels da imagem. Cada um desses valores terá uma banda em separado. Usaremos o método **ee.Image.constant()** para isso. Ele irá gerar 'n' bandas, respectivamente aos 'n' valores declarados pelo usuário. Caso somente um valor seja declarado, somente uma banda será criada.

var maskbands = ee.Image.constant(maskList)

Para criar a máscara, iremos comparar a banda BQA com cada uma dessas bandas que acabamos de criar. Para isso usaremos o método **eq(),** que executa um **teste de lógica binária**. Ele compara duas bandas e retorna valor 1 para pixels com valores idênticos e valor 0 para pixels com valores distintos. Assim, teremos 'n' bandas com valor 1 para bons pixels. O que faremos em seguida é usar um **redutor** para obter somente o valor máximo. Assim, combinaremos todas possibilidades de valores que

representam pixels válidos numa única camada. Por fim, renomearemos essa máscara com o nome ‘mask’.

var mask = qa.eq(maskbands).reduce('max').rename('mask')

Temos, portanto, nossa máscara. Seus pixels possuem valor ‘1’ e ‘0’. Quando a aplicarmos em uma imagem, ela irá descartar nessa imagem os pixels que coincidem com o valor ‘0’ e deixará somente os que coincidem com o valor ‘1’. Existe um método apropriado para isso: **updateMask()**. Adicionalmente, iremos selecionar da imagem mascarada somente as bandas B4, B3 e B2.

return image.updateMask(mask).select(['B4', 'B3', 'B2'])

Eis a nossa função ‘uniMask’, por completo:

```
function uniMask(image, maskList, qaband) {
   var qa = image.select(qaband)
   var maskbands = ee.Image.constant(maskList);
   var mask = qa.eq(maskbands).reduce('max').rename('mask')
   return image.updateMask(mask).select(['B4', 'B3', 'B2'])
   }
```

Agora, iremos declarar a lista de valores de pixels desejados e também o nome da banda indicadora de qualidade dos pixels para a coleção que iremos trabalhar:

```
var goodPix = ee.List([2720, 2724, 2728]);
var qaBand = ee.String('BQA');
```

Por fim, iremos aplicar essa função à uma coleção de imagens. Para isso, usaremos novamente o método map() (*vide* Script 3) para percorrer todas imagens da coleção e aplicar a função ‘uniMask’.

```
var collection = ee.ImageCollection('LANDSAT/LC08/C01/T1_TOA')
  .map(function(image) { return uniMask(image, goodPix, qaBand) })
```

Vamos adicionar ao mapa a coleção filtrada:

```
Map.addLayer(collection,'', 'Coleção Sem Nuvens')
```

Exercício 3: Altere a função uniMask para que ela calcule o NDVI da imagem e adicione o índice como uma banda adicional. Crie um gráfico para adicionar a série temporal compilada por dia do ano (ui.Chart.image.doySeries)

Script 5

```
// Construir função
function uniMask(image, maskList, qaband) {
  var qa = image.select(qaband) // select QA band
  var maskbands = ee.Image.constant(maskList)
  var mask = qa.eq(maskbands).reduce('max').rename('mask')
  return image.updateMask(mask).select(['B4', 'B3', 'B2'])
  Map.addLayer(mask, '', 'mask')
}

// Lista com valores de bons pixels
var goodPix = ee.List([2720, 2724, 2728]);    // LS8 C01 T1
// Nome da banda QA
var qaBand = ee.String('BQA');


// Aplicar a função numa coleção de imagens
var collection = ee.ImageCollection('LANDSAT/LC08/C01/T1_TOA')
  .map(function(image) {return uniMask(image, goodPix, qaBand)})

Map.addLayer(collection,'', 'Coleção Sem Nuvens')
```

Script 6 – Criando time-lapses com imagens de satélite

Em andamento....

Click

```
var lon = ui.Label();
var lat = ui.Label();

Map.onClick(function(coords)
{
     lon.setValue('lon: ' + coords.lon.toFixed(2)),
     lat.setValue('lat: ' + coords.lat.toFixed(2));

     var point = ee.Geometry.Point(coords.lon, coords.lat);
     var  chart  =  ui.Chart.image.series(ndvi.select('ndvi'),
point);

     chart.setOptions
     ({
     title: "Variação do NDVI no tempo",
     vAxis: {title: 'NDVI'},
     hAxis: {title: 'Data', format: 'MM-yy', gridLines: {count:
7}},
     });

     chart.style().set({
     position: 'bottom-right',
     width: '500px',
     height: '300px'
     });
     Map.add(chart);
})
```

Tópicos a adicionar no tutorial:

Combinando séries de diferentes sensores

Animações

Tabelas e vetores

Coleções interessantes:
‘WorldPop/POP’ - População mundial
'NOAA/DMSP-OLS/NIGHTTIME_LIGHTS' - Luzes noturnas

Dicas

Alguns atalhos que podem ajudar pelo caminho...

Para selecionar uma sequência de bandas, você pode usar: .select('B[1-5]'), em vez de .select('B1', 'B2', 'B3', 'B4', 'B5')

Respostas dos exercícios

Exercício 1

```
var srtm = ee.Image("USGS/SRTMGL1_003");
var declividade = ee.Terrain.slope(srtm)
Map.addLayer(declividade)
```

Referências

https://developers.google.com/earth-engine/tutorial_js_01

https://developers.google.com/earth-engine/tutorials#hands-on-intermediate-training

https://landsat.usgs.gov/sites/default/files/documents/Landsat8DataUsersHandbook.pdf