Kirti's Class

Hossein Noorazar

August 3, 2022

Contents

Chapter 1

Coding

Remark 1.0.1. *Use meaningful names. Refrain from using* `v`*, then double* `v`*;* `vv`*, and then* `w`*.*

Remark 1.0.2. *Refrain from copying your code from one file to another.*

When you copy your code from one file/module to another, it means you use it a lot. Thus, you can save it in a unique module and call it again and again. This is easier, faster, and more importantly sustainable. For example, if you plot a vector with your favorite settings, write a function once then use it. If later you need to change the font to be consistent with the journal you want to submit your paper to, then, you can change the font only once. Otherwise, you have to go to all the files you wrote before and change them one at a time! Just stop copy/pasting.

In Fig. 1.1 we see how a function is defined and documented in Python. A few points to make here using this example are given below.

```
def diversify(prices_arr, portfolio_size, invest_fund=1, preferred=None):
    """Return (stocks indices, weights [norm of eita - eita bar]^-1) for a diversified portfolio
    Arguments
    ---------
    prices_arr : array like
        Stock prices: `prices_arr.shape = (Ndays, Nstocks)`
    portfolio_size : int
        Number of diverse stocks to choose.
    preferred : list of int
        List of preferred vectors. These columns should be in the
        diversified return

    Returns
    -------
    weights_dict : dict(ind:weight)
        Dictionary of recommended diversification. Keys are the indices,
        and the values are the associated weights.

    stocks : array
        Recommended list of `portfolio_size` stocks for a diversified portfolio
    weights : array_like
        Array of weights of investment for each stock

    Example
    -------
    >>> prices_arr = np.arange(1, 1251).reshape((250, 5)) # for plotting
    >>> diversify(prices_arr=prices_arr, portfolio_size=3, invest_fund=1000)
    OrderedDict([(3, 450.15...), (1, 312.327...), (0, 237.52...)])
    """
```

Figure 1.1: Function Documentation.

Remarks about the function in Fig. 1.1;

Function Name is meaningful.

Input Variables have meaningful name.

First Input Variable There are different conventions. In this example we see the first input variable is named `prices_arr`. We see that not only the variable name is meaningful but also it is implying the type of the variable. It is an array. It is not a list. It is not a panda dataframe.

Variables with Default Values come last.

Function's Description Include the functions' descriptions the first thing in the function. The first line (text) can describe what the function does. Then, define the arguments; input variables. We have `prices_arr : array like` and then in the line below `Stock prices: 'prices_arr.shape = (Ndays, Nstocks)'`. So, immediately after the variable name its type is specified. In the line that follows we say what it is representing.

Include Example(s) You can include an example at the end of the description to show how the function is used. This is not necessary unless later you want to share your code, form a library and release it. Then, you can have pages like Python's documentation pages.

Make Comments throughout the code and functions. Write down what each line is doing or why. You can skip simple stuff. If you do not make comments and need to use the code after a month you may have hard time remembering.

The example above is on GitHub for you to review. The convention for documentation used in this example is the same as that of NumPy [1].

Once you write your functions in a central module you can import them into your driver modules and use them the way you import standard libraries/packages of Python or R.

Say we have two modules that are the engine of our project. One includes all the functions that generate stuff and the other includes all the plotting functions; `kirtis_class_core.py`, `kirtis_class_plot_core.py`. Say these two modules are saved in the following directory: `/Users/Documents/project1/`. Then we can import these modules into our driver and use them like so:

I use sublime text editor to write my codes

Algorithm 1.1: Import Your Python Modules.

```
0. import sys;
1. sys.path.append('/Users/Documents/project1/');
2. import kirtis_class_core as kcc ;
3. import kirtis_class_plot_core as kcpc ;
4. Use the function diversify from
   kirtis_class_core.py like
diversified_portfolio = kcc.diversify(x, y, z)
```

A simple code demonstrating this is on GitHub that you can look at; github.com/HNoorazar/Ag/tree/master/Kirtis_Class/Python.

In a similar way you can import your R modules into the drivers. Say your R module with all the functions in it is called `kirtis_class_core.R` located in `/Users/Documents/project1/`. You can import it like so in your driver:

Algorithm 1.2: Import Your R Modules.

```
1. source_path =
   "Users/Documents/project1/kirtis_class_core.R" ;
2. source(source_path) ;
3. Use the function diversify from R_engine_core.R
   like you use any function of R:
diversified_portfolio = diversify(x, y, z)
```

Remark 1.0.3. *Please note there are two standard conventions for variable names where the variable name is consist of more than one word. Either use an underscore or first letter of each word (starting from the second word) is capitalized;* `evi_ratio` *or* `eviRatio`.

Chapter 2

Keep it Organized

In each project do any or preferably all of the following. They may be inconvenient during the daily routine work. They, however, can save you a lot of time in rare events. Some events frequently occur where the price is low. Some events occurs rarely with a high price. An irrelevant example is "extreme weather events are low-probability high impact events" in power grid systems. In our case, there might come a time that you need to look back and figure out what you did and why you did it a few months ago. Keeping a *diary* can save you a lot of headache.

1. The easiest way of keeping your project flow organized is to include a numeric part in the name of files and folders. For example, `01_FetchData_from_GEE.py` can be the name of your first script with the same or similar name for the folder (`user/document/projectXYX/01_Data_from_GEE/`) for the data coming from this step; `Data_from_GEE.csv`.

   Then, `02_remove_noise.py`, `03_regression_smooth_data.py`, `04_plot_the_result.py`, `04_analyze_the_result.py`.

   Note that I have two items starting with "04_". These two are independent of each other and both depend on "`03_regression_smooth_data.py`".

2. Include dates in your codes. On top of the code document the dates and that helps you to see what you did last if you need to dig in the history and re-do stuff.

3. If you want to be on top of things, you can create something like Figs. 2.1 and 2.2. An end-to-end flowchart/doodle of the workflow.

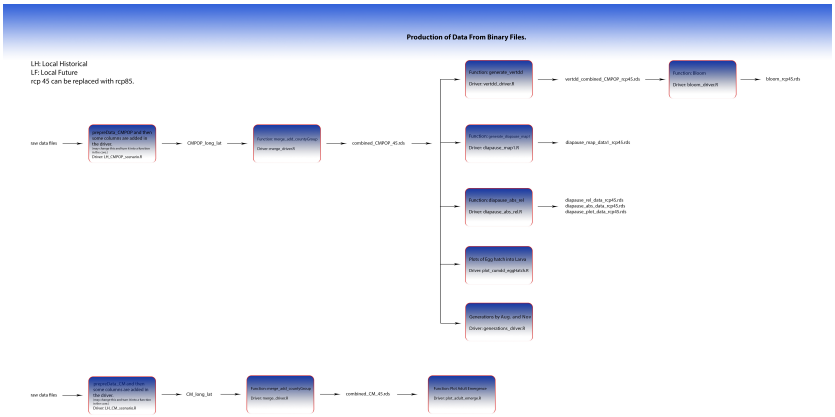


Figure 2.1: Workflow for one of my projects.

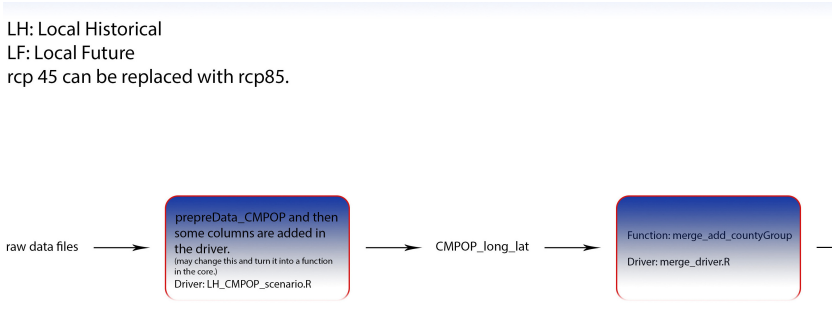


Figure 2.2: Workflow zoomed in.

Chapter 3

Google Earth Engine

3.1 Preface

Google Earth Engine sucks! Below (Fig. 3.1) we have a simple example to show GEE is very specific. Accessing to elements/entries of its object is not intuitive. Figuring out every single step is a challenge.

Figure 3.1: GEE sucks.

Here is the code used for generation of Fig. 3.1.

Algorithm 3.1: GEE Sucks.

```
1. print("Print #1:  3+10", 3+10) ;
2. var x=3 ;
3. var y=10 ;
4. print("Print #2:  x+y", x+y) ;

5. var big_delta_x = 3 ;
6. print("Print #3:  big_delta_x", big_delta_x) ;

7. var x_big = ee.List.sequence(-125.0, -111.3,
   big_delta_x) ;

8. print ("Print #4:  x_big", x_big) ;
9. print("Print #5:  x_big.get(1)", x_big.get(1)) ;
10. print("Print #6",
ee.Number(big_delta_x).add (ee.Number(x_big.get(1)))) ;

11. var aaa = x_big.get(1) + big_delta_x ;
12. // print("Print #7:  aaa", aaa) ;
13. print("Print #8:  ee.Number(aaa)",
   ee.Number(aaa)) ;
```

3.2 JavaScript or Python Interface

I think Python should be avoided in this particular case for the following reasons:

1. The interface is too slow,
2. The interface needs authentication every single time,
3. Google does not maintain the Python. Therefore, the functions are first written/updated for the JavaScript (JS) by Google, and the Python equivalents/updates will not be provided in a timely manner (who knows when?).
4. The tutorials for JS is already hard to find, it is much worse for Python. Again, since Google is responsible for JavaScript, it releases the tutorials for it, but not Python.

   P.S. tutorials for JS might be abundant, but finding your exact needs might be hard. Even when you find something you may not be sure if that is the best possible solution.

There are different products[1] that fall under different labels; tier 1 vs tier 2, collection 1 vs collection 2, level 1 and level 2. Some of these

[1] start here to collect some information. some of the products are deprecated and superseded and Google does not show them easily: here

have the same description on Google developer pages. For example, USGS Landsat 8 Surface Reflectance Tier 1 and USGS Landsat 8 Surface Reflectance Tier 2 have the same description and identical bands. In this particular example we want to use Tier 1. But we need a deeper understanding of differences(?)

Based on the information below and references therein, Collection 2 is an improvement over Collection 1[2]. It seems Collection-2 Level-2 Tier-1 should be the best, but in our plots it was not different from T1_SR (Fig. 3.2). Also keep in mind Collection-2 Level-2 bands must be scaled.

[2] Is there any time period for which Collection 2 does not exist but 1 does?

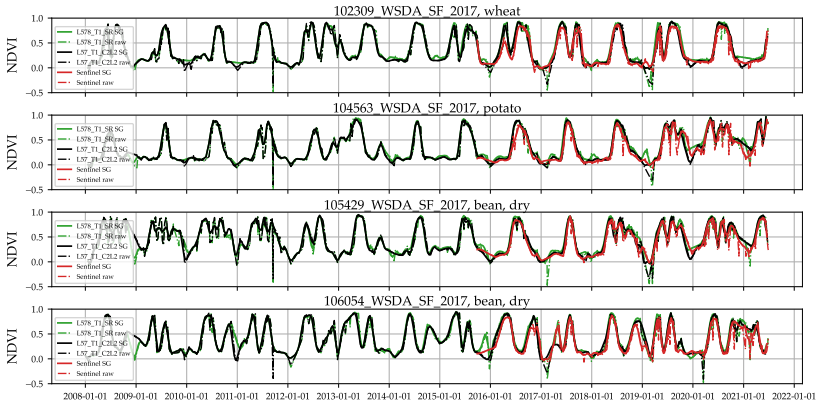


Figure 3.2: In this plot the data points from Landsat-5, -7, and -8 (Tier 1, Surface Reflectance, from GEE collection LANDSAT/LE07/C01/T1_SR) are merged together to form one vector. The same is done to Landsat-5 and -7 Collection-2 Level-2 (from GEE collection LANDSAT/LE07/C02/T1_L2). We can see they all are performing well.

Moreover, GEE [2] says "This dataset is the atmospherically corrected surface reflectance from the Landsat 7 ETM+ sensor." about "USGS Landsat 7 Surface Reflectance Tier 1" (LANDSAT/LE07/C01/T1_SR). On the other hand, it also says "Caution: This dataset has been superseded by LANDSAT/LC08/C02/T1_L2."

Collection-1 has only Level-1 data, however, Collection-2 has level-1 as well as Level-2.

Collection 1 Landsat Collection 1 was established in 2016 to improve archive management. Learn more about Collection 1 from the USGS.

Landsat Collection 1 consists of Level-1 data products generated from Landsat 8 Operational Land Imager (OLI)/Thermal Infrared Sensor (TIRS), Landsat 7 Enhanced Thematic Mapper Plus (ETM+), Landsat 4-5 Thematic Mapper (TM)*, and Landsat 1-5 Multispectral Scanner (MSS) instruments. **Collection 1 Tiers:**

Tier 1 "Landsat scenes with the highest available data quality are placed into Tier 1 and are considered suitable for time-series analysis." [3]

Tier 2 "Landsat scenes not meeting Tier 1 criteria during processing are assigned to Tier 2. Tier 2 scenes adhere to the same radiometric standard as Tier 1 scenes, but do not meet the Tier 1 geometry specification due to less accurate orbital information (specific to older Landsat sensors), significant cloud cover, insufficient ground control, or other factors." [3]

Collection 2 Landsat Collection 2 marks the second major reprocessing effort on the Landsat archive by the USGS that results in several data product improvements that harness recent advancements in data processing, algorithm development, and data access and distribution capabilities. Learn more about Collection 2 from the USGS.

Collection-2 Level-1 has different processings for different satellites [4]. It seems Collection-2 level-1 is TOA and Collection-2 level-2 is Surface Reflectance. "Collection-2 Level-2 science products are generated from Collection 2 Level-1 inputs that meet the <76 degrees Solar Zenith Angle constraint and include the required auxiliary data inputs to generate a scientifically viable product." [5]. "**Surface reflectance** (unitless) measures the fraction of incoming solar radiation that is reflected from the Earth's surface to the Landsat sensor. The LEDAPS and LaSRC surface reflectance algorithms correct for the temporally, spatially and spectrally varying scattering and absorbing effects of atmospheric gases, aerosols, and water vapor, which is necessary to reliably characterize the Earth's land surface." [5]. For the enhancement details please see [5].

3.3 Scaling the Bands

The purpose of this section is to make a point. Since it is an important point, a section is devoted to it.

If you look at the band tables on Sentinel-2, there is a column called *scale*. If you look at the band table of Landsat 8 Level 2, Collection 2, Tier 1, there are two columns called *scale* and *offset*. But such columns do not exist on Landsat 5 TM Collection 1 Tier 1 TOA Reflectance.

For some reason, Google Earth Engine has not scaled the bands and has made that your problem. So, you have to scale the bands properly during computations. If you forget to scale in case of Sentinel-2 and $NDVI = \frac{NIR-R}{NIR+R}$ you will be lucky since scales cancel out but that will not happen in case of EVI because of the additional 1 in the denomi-

nator (or in case of Landsat an off-set parameter is present as well);

$$\begin{aligned} EVI &:= G \times \frac{\rho_{NIR} - \rho_R}{\rho_{NIR} + C_1\rho_R - C_2\rho_B + L} \\ &= 2.5 \times \frac{\rho_{NIR} - \rho_R}{\rho_{NIR} + 6\rho_R - 7.5\rho_B + 1} \end{aligned} \tag{3.1}$$

Moreover, if you search the web for masking clouds in Sentinel, you will find the function maskS2clouds. If you look closely, in the last line the function is dividing the result by 10,000. Therefore, you do not have to scale the bands again in computation of VIs. However, you have to apply the `maskS2clouds` functions to the image collection before computing the VIs.

3.4 Access a Feature/Entry of a FeatureCollection

Suppose your `featurecollection` is called `SF`. In order to access its entries you have to convert it to a `list` and then use `get(.)`:

```
print ("SF.get(0)", SF.toList(4).get(0));
```

where `4` is the size of `SF` known in advance, and `0` is index of first entry of `SF`. In general you can use:

```
print ("SF.get(0)", SF.toList(SF.size()).get(index));
```

Please note if you use `SF.get(0)` you will get an error.

3.5 Add a Property to a Feature

Suppose you have uploaded a shapefile `SF` into your assets. The shapefiles usually have a component/slice called `data` (which is of type datatable) that can be accessed via `SF@data` in R. This component stores metadata corresponding to each polygon.

Say each polygon is an agricultural field that has some attributes associated with it such as irrigation type, area of the field, etc. After some computations on GEE you may want to attach these metadata to the output to use later. These metadata is referred to by `properties` on GEE. If you want to manually add a property to a feature you should use:

```
a_feature = a_feature.set('my_property', 1);
```

If you want to copy `properties` (metadata) of `feature_b` into `feature_a` you can do:

```
feature_a = feature_a.copyProperties(feature_b, ['ID', 'Irrigation_type']);
```

where `['ID', 'Irrigation_type']` is a subset of `properties` of `feature_b` to be copied into `feature_a`. I guess if that argument is dropped, then all `properties` will be copied.

3.6 Find Centroid of Polygons

Suppose you have a shapefile that you have uploaded to GEE as an *asset*. Here we will see how to find the centroids of the polygons in the shapefile. Let the name of shapefile be `Our_ShapeFile`. The function to compute centroids of the polygons in `Our_ShapeFile` is given by Alg. 4[3]. Line 4 of the Alg. 4 is keeping the columns of data slice in `Our_ShapeFile`; `Our_ShapeFile`@data.

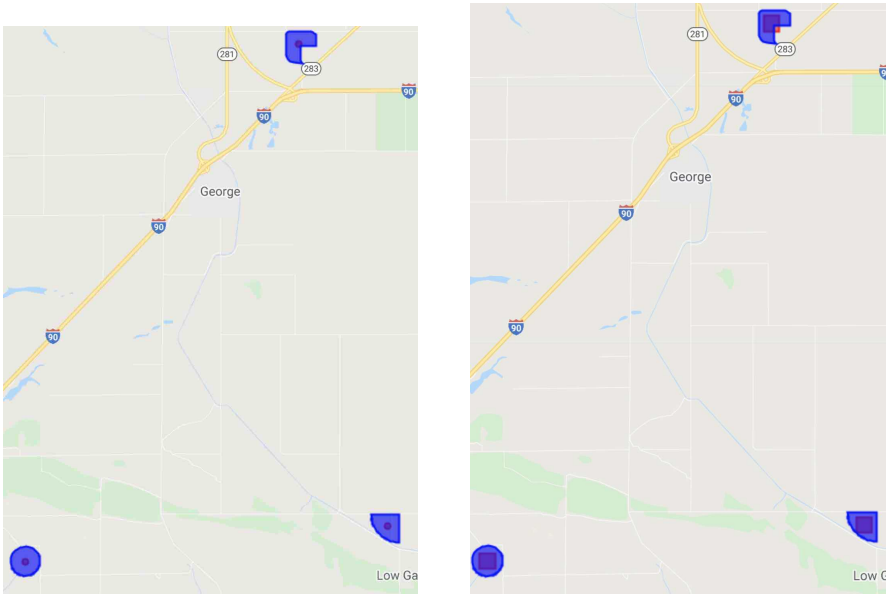
[3] This algorithm is accessible on GEE here.

Algorithm 3.2: Find Centroids of Polygons in a Shapefile.

```
1. function getCentroid(feature) {
    2. // Keep this list of properties.;
    3. var keepProperties = ['ID', 'county'];

    4. // Get the centroid of the feature's geometry.;
    5. var centroid = feature.geometry().centroid(); ;

    6. // Return a new Feature, copying properties from
the
        old Feature.;
    7. return ee.Feature(centroid).copyProperties
(feature,

keepProperties);
8. }

9. var SF = ee.FeatureCollection(Our_ShapeFile);
10. var centroids_from_GEE = SF.map(getCentroid);
```

Warning: Imagine your polygon looks like a doughnut (non-convex shape). Then the centroid would be in the center of the disk in the center of the doughnut which is not part of the doughnut/polygon/region of interest. So, if you want to look at an area around the centroid, then

that area (or parts of it, depending on how large the area is) would not belong to the polygon (See Fig. 3.3b; it is not a doughnut, but it delivers the message!)



(a) Bad Centroid

(b) Bad Buffer

Figure 3.3: Centroids and buffers around the centroids of polygons in a shapefile.

By adding one line (line 5.5 in Alg. 5) to the function `getCentroid(.)` we can get a buffer (a rectangular or a circle area) around the centroids.

Algorithm 3.3: Make a Buffer Around Centroids of Polygons.

```
1. function
 get_rectangle_arround_centroid(feature){
    2. // Keep this list of properties.;
    3. var keepProperties = ['ID', 'county'];

    4. // Get the centroid of the feature's geometry.;
    5. var centroid = feature.geometry().centroid(); ;
    5.5 centroid =
ee.Feature(centroid.buffer(200).bounds());
    6. // Return a new Feature, copying properties from the
    old Feature.;
    7. return ee.Feature(centroid).copyProperties(feature,

keepProperties);
8. }
9. var SF = ee.FeatureCollection(Our_ShapeFile);
10. var centroids_from_GEE =
                        SF.map
(get_rectangle_arround_centroid);
```

3.7 Cloud Filtering

Handling clouds for Sentinel and Landsat are different. Let us start by **Sentinel**.

First, the followings are equivalent:

- `var filtered = my_IC.filterMetadata( 'CLOUDY_PIXEL_PERCENTAGE', 'less_than', 70);`
- `var filtered = my_IC.filter('CLOUDY_PIXEL_PERCENTAGE < 70')`
- `var filtered = my_IC.filter(ee.Filter.lte('CLOUDY_PIXEL_PERCENTAGE', 70))`

They all filter out *images* with cloud cover less than or equal to 70%. Those images will NOT be in our `filtered` collection. Said differently, our `filtered` collection may include images that are covered by cloud up to 70%.

This is a pre-filtering step. Later, we can toss out the cloudy *pixels* from every single image.

Algorithm 3.4: Filter Cloudy Pixels for Sentinel.

```
1. function maskS2clouds(image) {

  2. // Each Sentinel-2 image has a bitmask band with
cloud
       mask information QA60.;
  3. var qa = image.select('QA60');

  4. // Bits 10 and 11 are clouds and cirrus,
respectively.;
  5. var cloudBitMask = 1 << 10 ;
  6. var cirrusBitMask = 1 << 11 ;

  7. // Both flags should be set to zero, indicating
clear
       conditions.;
  8. var mask = qa.bitwiseAnd(cloudBitMask)
.eq(0).and(
                qa.bitwiseAnd(cirrusBitMask) .eq(0));

  9. // Return the masked and scaled data, without
       the QA bands.
  10. return image.updateMask(mask)
                .divide(10000)
                .select("B.*")
     .copyProperties( image, ["system:time_start"]);
11. }
```

Note 1: Please note the last line in Alg. 6 is copying the system start time into the image which has nothing to do with clouds. It may be handy later.

Note 2: Please note the three (equivalent) pre-filtering of images mentioned above do not exist for Landsat!

Landsat(s) is a different satellite, and therefore, the cloud filtering must be handled differently; the band names that includes cloud information are different between Sentinel and Landsat or even among different Landsats.

Landsat-8 *Surface Reflectance* cloud mask [6]:

Algorithm 3.5: Filter Cloudy Pixels for Landsat-8 **Tier 1 and 2** *Surface Reflectance*.

```
function maskL8sr(image) {
  // Bits 3 and 5 are cloud shadow and cloud,
     respectively.;
  var cloudShadowBitMask = (1 << 3);
  var cloudsBitMask = (1 << 5);

  // Get the pixel QA band.;
  var qa = image.select('pixel_qa');

  // Both flags should be set to zero, indicating
clear
     conditions.;
  var mask = qa.bitwiseAnd(cloudShadowBitMask).eq(0)

.and(qa.bitwiseAnd(cloudsBitMask).eq(0));

  return image.updateMask(mask);
}
```

Note: This is written for Landsat-8 (Surface Reflectance Tier 1 and 2).

The code for masking the cloudy pixels in Landsat-4, 5, and 7 *Surface Reflectance* is given by [7] that is given below by Alg. 8:

Algorithm 3.6: Filter Cloudy Pixels for Landsat-4, 5, and 7 **Tier 1 and 2** *Surface Reflectance*.

```
1. function cloudMaskL457(image) {
2. var qa = image.select('pixel_qa');
3. // If the cloud bit (5) is set and the cloud confidence (7)
        is high or the cloud shadow bit is set (3),
        then it's a bad pixel.
4. var cloud = qa.bitwiseAnd(1 << 5)
                  .and(qa.bitwiseAnd(1 << 7))
                  .or(qa.bitwiseAnd(1 << 3));
5. // Remove edge pixels that don't occur in all bands
6. var mask2 = image.mask().reduce(ee.Reducer.min());
7. return image.updateMask(cloud.not()).updateMask(mask2);
10. }
```

I have copied the cloud masking functions from GEE development/data-product pages into a script that can be found here [8]. More on masking clouds of Sentinel-2 and shadows are provided here [9] by GEE developers.

Another way of masking cloud used in [10]:

Algorithm 3.7: Filter Cloudy Pixels for Landsat-7 and 8 *TOA*; LANDSAT/LC08/C01/T1_TOA and LANDSAT/LE07/C01/T1_TOA.

```
1. function cloudMask(image) {
2. var cloudscore = ee.Algorithms.Landsat
                        .simpleCloudScore(image).
                        .select('cloud');
3. return image.updateMask(cloudscore.lt(50));
10. }
```

3.8 Timelines

Figure 3.4 shows the timeline of Landsat satellites [11] and Table 3.1 shows the exact dates.

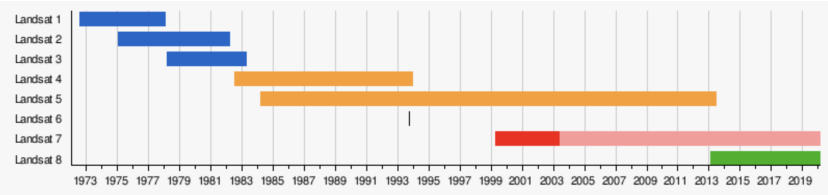


Figure 3.4: Landsat Timeline.

Satellite	Launched	Terminated
Landsat 5	1 March 1984	5 June 2013
Landsat 6	5 October 1993	5 October 1993
Landsat 7	15 April 1999	Still active
Landsat 8	11 February 2013	Still active
Landsat 9	16 September 2021 (planned)	-

Table 3.1: Landsat timeline table.

3.9 Band Names and Indices

Band names are different in each instrument (see Table 3.2). Hence the indices must be defined differently using proper band names. Below we see some of indices. Table 3.3 also provides more insight about the bandwidths of the satellites. The bandwidths are very similar. If their minimal differences makes any difference I am not aware of it and do not care. Go nuts if you wish; figure out why, what, how. Bandwidths of Sentinel-2 is found on Wikipedia [4] and Bandwidths of Landsats can be found on GEE pages (e.g. [2]).

[4] Sentinel Bandwidths. https://en.wikipedia.org/wiki/Sentinel-2. [Online; accessed August 3, 2022]

$$
\begin{aligned}
EVI &= G \times \frac{NIR - R}{NIR + C1 \times R - C2 \times B + L} \\
EVI_S &= 2.5 \times \frac{B8 - B4}{B8 + 6 \times B4 - 7.5 \times B2 + 1} \\
EVI_8 &= 2.5 \times \frac{B5 - B4}{B5 + 6 \times B4 - 7.5 \times B2 + 1} \\
EVI_7 &= 2.5 \times \frac{B4 - B3}{B4 + 6 \times B3 - 7.5 \times B1 + 1}
\end{aligned}
\tag{3.2}
$$

where NIR is near infrared, R is Red, B is blue, EVI_8 is the Enhanced Vegetation Index (EVI) in Landsat-8 [13], and EVI_S is the EVI in Sentinel; The NIR band in Landsat-8 is $B5$ [14] and for Sentinel is $B8$.

"EVI is similar to Normalized Difference Vegetation Index (NDVI) and can be used to quantify vegetation greenness. However, EVI cor-

Satellite	NIR	Red	Blue
Sentinel	B8	B4	B2
Landsat-8	B5	B4	B2
Landsat-7	B4	B3	B1
Landsat-5	B4	B3	B1

Table 3.2: Some Band Names in Satellites.

Satellite	NIR	Red	Blue
Sentinel-2A	B8: 0.77 – 0.88 μm	B4: 0.65 – 0.68 μm	B2: 0.46 – 0.52 μm
Sentinel-2B	B8: 0.78 – 0.88 μm	B4: 0.65 – 0.68 μm	B2: 0.46 – 0.52 μm
Landsat-8	B5: 0.85 – 0.88 μm	B4: 0.64 – 0.67 μm	B2: 0.45 – 0.51 μm
Landsat-7	B4: 0.77 – 0.90 μm	B3: 0.63 – 0.69 μm	B1: 0.45 – 0.52 μm
Landsat-5	B4: 0.77 – 0.90 μm	B3: 0.63 – 0.69 μm	B1: 0.45 – 0.52 μm
Landsat-7 C2 L2	SR_B4: 0.77 – 0.90 μm	SR_B3: 0.63 – 0.69 μm	SR_B1: 0.45 – 0.52 μm
Landsat-5 C2 L2	SR_B4: 0.77 – 0.90 μm	SR_B3: 0.63 – 0.69 μm	SR_B1: 0.45 – 0.52 μm

Table 3.3: Some Band Wavelengths. The bandwidths are very similar. If their minimal differences makes any difference I am not aware of it and do not care. Go nuts if you wish; figure out why, what, how.

rects for some atmospheric conditions and canopy background noise and is more sensitive in areas with dense vegetation. It incorporates an "L" value to adjust for canopy background, "C" values as coefficients for atmospheric resistance, and values from the blue band (B). These enhancements allow for index calculation as a ratio between the R and NIR values, while reducing the background noise, atmospheric noise, and saturation in most cases" [13].

Below are the NDVIs for Landsat-4 to Landsat-7 [15], Landsat-8 [15], and Sentinel:

$$\begin{aligned} NDVI &= \frac{NIR - R}{NIR + R} \\ NDVI_S &= \frac{B5 - B4}{B5 + B4} \\ NDVI_8 &= \frac{B8 - B4}{B8 + B4} \\ NDVI_{4-7} &= \frac{B4 - B3}{B4 + B3} \end{aligned} \tag{3.3}$$

Landsat-7 has 8-day NDVI composite already provided by GEE [16]. This product is based on TOA data which is not perfect! However, it seems running some smoothing methods on it can make it useful.

3.10 Tiny Tips, Big Problems

The tips in this section are useful for beginners and if you want to do something that is unusual.

Some times you may find yourself in a situation for which you are using the biggest sledgehammer to deal with the tiniest nail. In these scenarios the empire of Google does not have a function (for good reasons most likely) to do the job. If brute force is the chosen approach then these tips may be handy. If you are the only person on the planet who wants to do a certain thing, maybe you need to think again, and let go of useless approaches.

Object Types There are two types of objects or functions. Some are called server-side. Some are called client-side. Here is an example

that shows a client-side object does not work with server-side object.

It is strongly advised to avoid using/writing client-side objects/functions. The client-side objects also make the server/code/interface be very slow, freeze at times.

Batch Export This is an example that Google does not think is useful. But if you need to export a collection of images you can do it either using a for-loop for which you may need to look at the previous example. Or, you can use `batch.Download.ImageCollection.toDrive(.)`. Both of these approaches are demonstrated here.

Two remarks in this regard. First, the function for downloading the image collection as a batch[5] behaves strangely.[6] In Fig. 3.5 there are 4 parts. The top left shows two images in a folder; one is exported via for-loop and the other is exported via batch-download. In the batch-downloaded image, naked eye cannot see anything, it is black and white. After opening it, it turns all into white (lower left). But the image exported via for-loop can be seen with naked eye (top right). The strange event is that the batch-downloaded image, can be seen if it is opened via Python or GIS (lower right image)!

[5] `batch.Download.ImageCollection.toDrive(.)`

[6] I was visualizing the images as RGB images and exporting them; `var imageRGB = an_image.visualize(vizParams)`. I am not too sure if the batch download's problem is specific to RGB images.

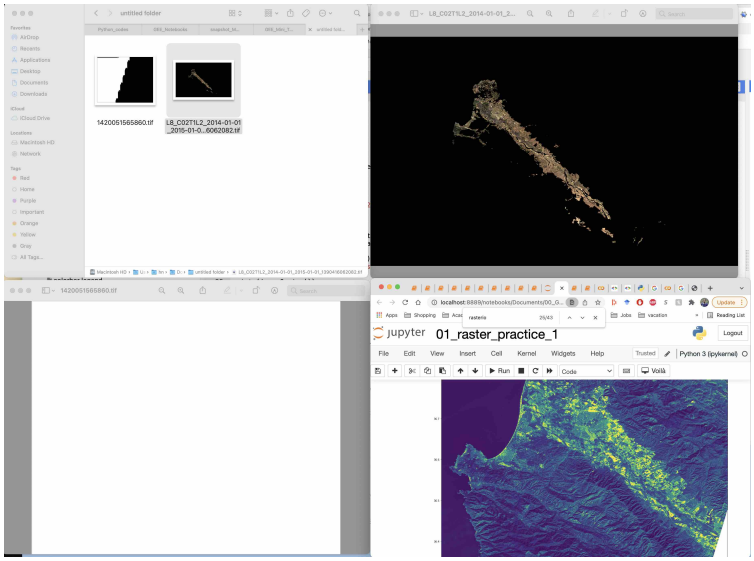


Figure 3.5: Strange Behavior of Batch-Download.

The images I exported turned out to be black and white. Secondly, any time a data is exported on GEE interface, you need to click on the **Run** button on **Task** tab. Perhaps Python can be used to avoid this problem, as well as server-side/client-side problem altogether.

Enough is a Feast Only choose the bands you need. The reason is that when you apply an operation to the image collections, it is applied to all bands. For example, you may want to take maximum value of NDVI per pixel in an image collection. When you use

`qualityMosaic(.)` function, it does not know your intention. It is applied to all bands. This can be expensive and take time. Moreover, suppose you merge data from Landsat 5, 7, and 8 on Google Earth Engine to use `qualityMosaic(.)` on the merged collection. Some additional bands are present in Landsat 8 that are not present in Landsat 5. Therefore, `qualityMosaic(.)` will not know what to do with those bands. Therefore, after 3 days of waiting you will get an error!

3.11 Full Code Examples; Landsat and Sentinel

Here are two examples, one for Landsat-8 [17] and one for Sentinel-2 [18]. They are both on the GEE Mini Tutorial repo on Google Earth Engine [19]. I have had problems with sharing repo in the past. If that does not work, you can copy the codes from the GitHub repo where this PDF is located at [20]. There is also a shapefile on Google drive [21] that is used in some of these codes.

3.11.1 *Code Example*

Merging data on GEE; see appendix of the paper
https://www.mdpi.com/2072-4292/11/7/820/htm

3.12 Remarks

Filter Clouds and Scale. Please look at the way clouds and shadows are filtered and proper bands are scaled with not hard-coding on the GEE page[7] in the function `prepSrL8(.)`.

Merge on GEE You can merge two image collections (e.g. image collections from Landsat 7 and 8) on GEE like so

```
var merge_IC = ee.ImageCollection(col_1.merge(col_2));
```

[7] Compositing and Mosaicking. https://developers.google.com/earth-engine/guides/ic_composite_mosaic. [Online; accessed August 3, 2022]

3.13 Beyond GEE

I like to advice once you are done with GEE, read your CSV files (Python/R/etc.) and round the digits to 2 (or 3?) decimal places if you will. That reduces the file sizes. Of course this matters when you are working with substantial number of fields.

Bibliography

[1] Style guide. https://numpydoc.readthedocs.io/en/latest/format.html. [Online; accessed August 3, 2022].

[2] Landsat-7 Tier-1 Surface Reflectance Bandwidths. https://developers.google.com/earth-engine/datasets/catalog/LANDSAT_LE07_C01_T1_SR?hl=da. [Online; accessed August 3, 2022].

[3] Landsat collection-1 description. https://www.usgs.gov/core-science-systems/nli/landsat/landsat-collection-1?qt-science_support_page_related_con=1#qt-science_support_page_related_con. [Online; accessed August 3, 2022].

[4] Collection-2 Level-1 Description. https://www.usgs.gov/core-science-systems/nli/landsat/landsat-collection-2-level-1-data?qt-science_support_page_related_con=1#qt-science_support_page_related_con. [Online; accessed August 3, 2022].

[5] Collection-2 Level-2 Description. https://www.usgs.gov/core-science-systems/nli/landsat/landsat-collection-2-level-2-science-products. [Online; accessed August 3, 2022].

[6] Landsat-8 cloud mask. https://developers.google.com/earth-engine/datasets/catalog/LANDSAT_LC08_C01_T2_SR#bands. [Online; accessed August 3, 2022].

[7] Cloud cleaning in Landsat4-7. https://developers.google.com/earth-engine/datasets/catalog/LANDSAT_LE07_C01_T2_SR. [Online; accessed August 3, 2022].

[8] Cloud masking functions on GEE - a part of this mini-tutorial. https://code.earthengine.google.com/?scriptPath=users%2Fhnoorazar%2FGEE_Mini_Tutorial%3ACloudMaskings. [Online; accessed August 3, 2022].

[9] Cloud and shadow masking in Sentinel. https://developers.google.com/earth-engine/tutorials/community/sentinel-2-s2cloudless. [Online; accessed August 3, 2022].

[10] Haifeng Tian, Ni Huang, Zheng Niu, Yuchu Qin, Jie Pei, and Jian Wang. Mapping Winter Crops in China with Multi-Source Satellite Imagery and Phenology-Based Algorithm. *Remote Sensing*, 11(7), 2019.

[11] Landsat timelines from Wikipedia. https://en.wikipedia.org/wiki/Landsat_program. [Online; accessed August 3, 2022].

[12] Sentinel Bandwidths. https://en.wikipedia.org/wiki/Sentinel-2. [Online; accessed August 3, 2022].

[13] Landsat-8 EVI. https://www.usgs.gov/core-science-systems/nli/landsat/landsat-enhanced-vegetation-index?qt-science_support_page_related_con=0#qt-science_support_page_related_con. [Online; accessed August 3, 2022].

[14] Landsat-8 band names. https://www.usgs.gov/faqs/what-are-band-designations-landsat-satellites?qt-news_science_products=0#qt-news_science_products. [Online; accessed August 3, 2022].

[15] Landsat-4 NDVI. https://www.usgs.gov/core-science-systems/nli/landsat/landsat-normalized-difference-vegetation-index?qt-science_support_page_related_con=0#qt-science_support_page_related_con. [Online; accessed August 3, 2022].

[16] Landsat-7 8-day NDVI composite. https://developers.google.com/earth-engine/datasets/catalog/LANDSAT_LE07_C01_T1_8DAY_NDVI. [Online; accessed August 3, 2022].

[17] Full Code Example; Landsat-8. https://code.earthengine.google.com/?scriptPath=users%2Fhnoorazar%2FGEE_Mini_Tutorial%3ALandsat8_C2L2_NDVI. [Online; accessed August 3, 2022].

[18] Full Code Example; Sentinel-2. https://code.earthengine.google.com/?scriptPath=users%2Fhnoorazar%2FGEE_Mini_Tutorial%3ASentinel_NDVI. [Online; accessed August 3, 2022].

[19] GEE Mini Tutorial Repo on Earth Engine. https://code.earthengine.google.com/?accept_repo=users/

hnoorazar/GEE_Mini_Tutorial. [Online; accessed August 3, 2022].

[20] Mini Tutorial on Github. https://github.com/HNoorazar/Ag/tree/master/remote_sensing_for_others. [Online; accessed August 3, 2022].

[21] Shapefile on Google Drive. https://drive.google.com/drive/folders/11X_NCDw8ZKVR7dNa2j_8wEU_VQ5oOgPd?usp=sharing. [Online; accessed August 3, 2022].

[22] Compositing and Mosaicking. https://developers.google.com/earth-engine/guides/ic_composite_mosaic. [Online; accessed August 3, 2022].