Есть класс который используется в увеличении изображений датасета с сохранением геопривязки

class combiner():

'''combine npblock and geodata'''

def \_\_init\_\_(self, rasters: list):

для него реализовать:

1. процедуру (последовательность функций) positioning, выявляющую соотнесение индекса p\* в имени файла (rasters[k].abs) и пары индексов (i, j) обозначающих положение файла, соответствующего индексу относительно других на прямоугольной / квадратной матрице.

Пример: p30, p31, p32 → (30, (0, 0)), (31 (0, 1)) , (32, (0, 2)). # структура может быть другая.

Идея реализации: сортировка списка [index, (bound\_1, bound\_2 …?)] , затем проверка — идут ли bound\_1, bound\_2 … по порядку или индекс пропущен. Может быть другая. Хорошая процедура избавит от использования попарных проверок is\_vertival\_heighbours.

~~Идея 2. Найти ширину и последовательно увеличивать индексы.~~

2. вычисление булевого параметра self.has\_gap — True, если есть хотя бы один nan — блок, т. е. недостающий индекс.

3. вычисление размера высоты и ширины итоговой матрицы в пикселях (может быть не квадтратной).

~~4. предложить способ определения~~ **~~нового~~** ~~индекса p\* - отказываемся от суффикса \_seg\_ (при разделении) в имени файла (потому что тогда надо вводить суффикс и для объединения), т.е. изменем индекс изображения.~~  Класс только связывает npblock и geodata, соответственно вызывающая функция отвечает за индекс.

5. вычисление параметра transform матрицы объединения.

6\* записать датасет с изображением размера 1024\*1024 пикселя без пересечений. Для чего решить задачу поиска нового индекса, объединяющего исходные, с сохранением логики, т. е. индексы идут по порядку слева направо.

Есть класс который используется в уменьшении изображений датасета с сохранением геопривязки

class separator():

'''separate file to num\_npblocks and geodata'''

def \_\_init\_\_(self, file: file\_design, num\_npblocks: int):

для него реализовать:

0 (связано с 3.). проверку корректности аргумента num\_npblocks — можно ли разделить на равные квадраты. (могут быть нечетные длины этих квадратов)

1. процедуру get\_np((i, j)) возвращающую матрицу значений сегмента, i \* j <= num\_npblocks, i — row, j — column.

2. процедуру get\_transform((i, j)) возвращающую матрицу rasterio.transform.Affine значений сегмента.

3. вычисление размера матрицы сегмента в пикселях (только квадратная)

Примеры плохого кода, делающие преобразование transform

def get\_segmented\_transform(transform: affine.Affine, output\_image\_edge: int, slice\_number: int ):

segments\_in\_row = 128 / output\_image\_edge

transform\_as\_ndarray = np.array(transform)

t = transform\_as\_ndarray

x\_in\_row = int(slice\_number % segments\_in\_row)

y\_in\_col = int(slice\_number / segments\_in\_row)

x\_position = t[2] + output\_image\_edge \* t[0] \* x\_in\_row

y\_position = t[5] + output\_image\_edge \* t[4] \* y\_in\_col

return rasterio.transform.Affine(t[0] , t[1], x\_position, t[3], t[4], y\_position)

def get\_scaled\_transform(transform: affine.Affine, output\_image\_edge: int):

scale\_factor = 128 / output\_image\_edge

transform\_as\_ndarray = np.array(transform)

t = transform\_as\_ndarray

return rasterio.transform.Affine(t[0] \* scale\_factor, t[1], t[2], t[3], t[4] \* scale\_factor, t[5])