SoilGrids250 — Import tiles via WCS

D G Rossiter

david.rossiter@isric.org

12-January-2025

Table of Contents

Introduction

SoilGrids is a system for global digital soil mapping that makes use of global soil profile information and covariate data to model the spatial distribution of soil properties across the globe. SoilGrids250 is a collections of soil property maps at six standard depths at 250 m grid resolution.

SoilGrids250 filenames, procedures etc. are explained in a FAQ. The choice of the Goode Homolosine projection is explained in Moreira de Sousa, L., L. Poggio, and B. Kempen. 2019. *Comparison of FOSS4G Supported Equal-Area Projections Using Discrete Distortion Indicatrices. ISPRS International Journal of Geo-Information* 8(8): 351. https://doi.org/10.3390/ijgi8080351.

Download of SoilGrids layers is most convenient by Web Coverage Service (WCS), see this description of procedures from ISRIC.

This script creates a tile for a property and depth slice, over a Area of Interest delimited by geographic coördinates. Tiles are created in EPSG 4326 (WGS84 long/lat) with SoilGrid's nominal 250 m grid resolution.

This tile can then be compared with other PSM products or combined into a raster stack for pattern analyis.

To use this script:

1. Adjust the directory structure to your system.

Steps 2–3 refer to the YAML headers of the R Markdown source document, or external calls with `knitr::render`. You can ask to be prompted for the parameters with the R command at the console:

```
rmarkdown::render("SoilGrids250_WCS_import.Rmd", output_format =
"html_document",
                  params = "ask")
```

or you can specify them directly and run this script from the R console, e.g.,

```
rmarkdown::render("SoilGrids250_WCS_import.Rmd", output_format = NULL,
                    params = list(lrc_long = -76, lrc_lat = 42,
                                  size = 1, quantile.n = 4,
                                  voi.n = 3, depth.n = 1))
```

Here we specify `output_format = NULL` to just get the tile but not produce a document.

See the respective sections of the code for the parameter definitions.

2. Select a property and quantile and a depth slice, using the YAML header or by knitting with parameters.

3. Select an Area of Interest, typically a 1 × 1° tile, using the YAML header or by knitting with parameters.

4. Either compile to HTML or PDF ("knit"), or "Run All" within R Markdown, or call `rmarkdown::render` from the R command line, as explained above.

5. The processed tile will be in the directory structure, in a subdirectory named for the AOI.

Directories

Set base directories, specific to the local file system.

1. `base.dir.import`: This is where downloaded large GeoTIFF are located. Because of their size they may be on a separate file system, e.g., removable or networked drive.

2. `base.dir`: This is where the processed SoilGrids250 maps are stored.

```
(base.dir.sg <- path.expand("~/ds_reference/DSM2025/"))

[1] "/Users/rossiter/ds_reference/DSM2025/"
```

```
(base.dir.sg.import <- path.expand("~/tmp/DSM2025/"))

[1] "/Users/rossiter/tmp/DSM2025/"
```

These are the base of destination directories built below

Packages

```
library(sf)              # spatial data types
library(terra)           # raster data, replaces `raster`
require(rgdal)           # directly access GDAL functions -- deprecated but
still works
require(gdalUtilities)
require(XML)
```

GDAL is used for spatial data import/export, coördinate systems etc. Check the GDAL installation, and whether the WCS service is available.

```
gdal() # version
gdal.drivers <- terra::gdal(drivers = TRUE)
ix <- which(gdal.drivers$name  == "WCS")
gdal.drivers[ix, ]
```

Coördinate Reference System (CRS)

We want to use geographic coördinates for the tile. But the ISRIC WCS does not seem to serve this – or at least, I can not figure out how. So, we must download SoilGrids250 in the native Homolosine CRS. For this we need to know the bounding box in that CRS.

Note: This CRS with pseudo-EPSG code 152160 should be added to to the epsg file of the PROJ database on your system [1] as a final line, as explained here. But for now do not do this, just specify the projection directly.

```
crs.igh <- '+proj=igh +lat_0=0 +lon_0=0 +datum=WGS84 +units=m +no_defs'
```

Parameters

Define the variables for the soil property and layer of interest. See here for the naming conventions

`Q0.05` - 5% quantile from the Quantile Random Forest (QRF); `Q0.5` - median of the distribution from the QRF – note *not* `Q0.50`; `Q0.95` - 95% quantile from the QRF; `mean` - mean of the distribution.

```
quantile.list <- c("Q0.05", "Q0.5", "Q0.95", "mean")
```

---

[1] for example `/Library/Frameworks/PROJ.framework/Versions/6/Resources/proj/epsg`

Here are the properties predicted by SoilGrids250:

```
voi.list.sg <- c("clay", "silt", "sand", "phh2o", "cec", "soc", "bdod",
"cfvo", "nitrogen", "ocd")
```

We do not include ocs (Organic C stocks) here because that is for the whole profile, not by depth slice.

Here are the depth slices predicted by SoilGrids250:

```
depth.list <- paste0(c("0-5", "5-15", "15-30", "30-60", "60-100", "100-
200"),"cm")
```

Parameters for this run:

```
print(paste("VOI:", voi.list.sg[params$voi.n]))

[1] "VOI: cfvo"

print(paste("Depth slice:", depth.list[params$depth.n]))

[1] "Depth slice: 0-5cm"

print(paste("Quantile:", quantile.list[params$quantile.n]))

[1] "Quantile: mean"
```

Set the property, depth and quantile from the YAML or rendering parameters:

```
voi <- voi.list.sg[params$voi.n]
depth <- depth.list[params$depth.n]
quantile.sg <- quantile.list[params$quantile.n]
(voi_layer <- paste(voi, depth, quantile.sg, sep="_"))

[1] "cfvo_0-5cm_mean"
```

Area of Interest (AOI)

The AOI is a square tile using WGS84 geographic coördinates. A $1 \times 1°$ is typical.

Specify the *lower-right corner* and *tile size* from the YAML or rendering parameters:

```
tile.lrc <- c(params$lrc_long, params$lrc_lat) # lower-right corner
size <- params$size                # tile dimensions
print(paste("Lower-right corner: longitude", params$lrc_long,
            "; latitude", params$lrc_lat,
            "; tile size:", params$size, "degree square"))

[1] "Lower-right corner: longitude 80 ; latitude 21 ; tile size: 1 degree
square"
```

Compute the four corner and the bounding box. Note because of the projection this is somewhat larger than a $1 \times 1°$ tile.

```
tile.ulc <- c(tile.lrc[1]-size, tile.lrc[2]+size)   # upper-left corner
m <- matrix(c(tile.lrc[1]-size, tile.lrc[2]+size,  #ulc
              tile.lrc[1], tile.lrc[2]+size,  #urc
              tile.lrc[1], tile.lrc[2],       #lrc
              tile.lrc[1]-size, tile.lrc[2]  #lcc
              ),
            nrow=4, byrow = TRUE)
m <- rbind(m, m[1,]) # close the polygon
bb.ll <- st_sfc(st_polygon(list(m)))
```

The CRS of this bounding box is WGS84 geographic coördinates (EPSG code 4326).

```
st_crs(bb.ll) <- 4326
# print(bb.ll)
# st_boundary(bb.ll)
```

A prefix for directories, to keep AOI results separate.

```
(AOI.dir.prefix <- paste0("lat", tile.lrc[2], tile.ulc[2],
                          "_lon", tile.ulc[1], tile.lrc[1]))

[1] "lat2122_lon7980"
```

Destination directories

Set import and destination directories, adding to the base directories the variable of interest, quantile, depth. Make sure the directory exists.

```
(dest.dir.sg.import <-  paste0(base.dir.sg.import, AOI.dir.prefix))

[1] "/Users/rossiter/tmp/DSM2025/lat2122_lon7980"

if (!dir.exists(dest.dir.sg.import)) {
   dir.create(dest.dir.sg.import, recursive = TRUE)
}
(dest.dir.sg <-  paste0(base.dir.sg,
                        AOI.dir.prefix))

[1] "/Users/rossiter/ds_reference/DSM2025/lat2122_lon7980"

if (!dir.exists(dest.dir.sg)) {
  dir.create(dest.dir.sg, recursive = TRUE)
}
```

Convert the long/lat bounding box to the SoilGrids250 projection. We want the extreme values in both X and Y, to ensure we cover the whole tile. If we just use the corners we will cut off some parts at the upper-right an lower-left.

```
(bb.igh <- st_transform(bb.ll, crs.igh))         # reproject the polygon
```

```
Geometry set for 1 feature
Geometry type: POLYGON
Dimension:     XY
Bounding box:  xmin: 8397053 ymin: 2337709 xmax: 8535870 ymax: 2449029
Projected CRS: +proj=igh +lat_0=0 +lon_0=0 +datum=WGS84 +units=m +no_defs
```

```
(bb.igh.coords <-  st_coordinates(bb.igh)[,1:2]) # convert to coördinates, we
only need 2D
```

```
           X       Y
[1,] 8397053 2449029
[2,] 8500266 2449029
[3,] 8535870 2337709
[4,] 8431944 2337709
[5,] 8397053 2449029
```

```
#  convert to a bounding box, must order these as c(ulx, uly, lrx, lry)
bb.sg <- as.vector(c(min(bb.igh.coords[,"X"]),
                     max(bb.igh.coords[,"Y"]),
                     max(bb.igh.coords[,"X"]),
                     min(bb.igh.coords[,"Y"])))
```

Accessing SoilGrids250 WCS with GDAL

Adapted from instructions by Luis de Souza (ISRIC), see https://git.wur.nl/isric/soilgrids/soilgrids.notebooks/-/blob/master/markdown/wcs_from_R.md.

The idea is to set up an XML file with the request and then feed that to GDAL commands.

Note on the XML file name: re-using the same file name if this is called from another script seems to somehow not refresh a server cache – unclear how this happens – but anyway, this is why we make each XML file with a temporary name.

```
wcs_request <- "DescribeCoverage"
wcs_path <- paste0("https://maps.isric.org/mapserv?map=/map/",voi,".map") #
Path to the WCS. See maps.isric.org
wcs_service = "SERVICE=WCS"
wcs_version = "VERSION=2.0.1"
(wcs <- paste(wcs_path, wcs_service, wcs_version, wcs_request, sep="&"))
```

```
[1]
"https://maps.isric.org/mapserv?map=/map/cfvo.map&SERVICE=WCS&VERSION=2.0.1&D
escribeCoverage"
```

```
l1 <- newXMLNode("WCS_GDAL")
l1.s <- newXMLNode("ServiceURL", wcs, parent=l1)
l1.l <- newXMLNode("CoverageName", voi_layer, parent=l1)
# Save to local disk
xml.out <- paste0(".", tempfile("sg","",".xml"))
```

```r
tmp <- saveXML(l1, file = xml.out)
gdalinfo(xml.out)
```

```
Driver: WCS/OGC Web Coverage Service
Files: ./sg165a7cf630b7.xml
Size is 159246, 58034
Coordinate System is:
PROJCRS["Interrupted_Goode_Homolosine",
    BASEGEOGCRS["WGS 84",
        DATUM["World Geodetic System 1984",
            ELLIPSOID["WGS 84",6378137,298.257223563,
                LENGTHUNIT["metre",1]],
            ID["EPSG",6326]],
        PRIMEM["Greenwich",0,
            ANGLEUNIT["Degree",0.0174532925199433]]],
    CONVERSION["unnamed",
        METHOD["Interrupted Goode Homolosine"]],
    CS[Cartesian,2],
        AXIS["(E)",east,
            ORDER[1],
            LENGTHUNIT["metre",1,
                ID["EPSG",9001]]],
        AXIS["(N)",north,
            ORDER[2],
            LENGTHUNIT["metre",1,
                ID["EPSG",9001]]]]
Data axis to CRS axis mapping: 1,2
Origin = (-19949875.000000000000000,8361125.000000000000000)
Pixel Size = (250.000000000000000,-250.000000000000000)
Corner Coordinates:
Upper Left  (-19949875.000, 8361125.000)
Lower Left  (-19949875.000,-6147375.000)
Upper Right (19861625.000, 8361125.000)
Lower Right (19861625.000,-6147375.000)
Center      (   -44125.000, 1106875.000) (  0d51'35.71"W,  9d56'35.62"N)
Band 1 Block=1024x512 Type=Int16, ColorInterp=Undefined
  NoData Value=-32768
  Overviews: 79623x29017, 39811x14508, 19905x7254, 9952x3627, 4976x1813,
2488x906, 1244x453, 622x226
```

```r
file.remove(xml.out)
```

```
[1] TRUE
```

```r
# also remove 'helper' XML if created by saveXML
xml.DC <- paste0(".", strsplit(xml.out,".", fixed = TRUE)[[1]][2], ".DC.xml")
if (file.exists(xml.DC)) file.remove(xml.DC)
```

```
[1] TRUE
```

Ignore the GDAL warnings, because we know the Homolosine projection used in SG250 is not in the GDAL database.

Get the coverage:

```
(wcs <- paste(wcs_path,wcs_service,wcs_version,sep="&"))

[1]
"https://maps.isric.org/mapserv?map=/map/cfvo.map&SERVICE=WCS&VERSION=2.0.1"

l1 <- newXMLNode("WCS_GDAL")
l1.s <- newXMLNode("ServiceURL", wcs, parent=l1)
l1.l <- newXMLNode("CoverageName", voi_layer, parent=l1)
xml.out <- paste0(".", tempfile("sg","",".xml"))
tmp <- saveXML(l1, file = xml.out)
(file.out <- paste0(dest.dir.sg.import, "/", voi_layer, '.tif'))

[1] "/Users/rossiter/tmp/DSM2025/lat2122_lon7980/cfvo_0-5cm_mean.tif"

gdal_translate(xml.out, file.out,
    tr=c(250,250),
    projwin = bb.sg, projwin_srs = crs.igh, # corners in this CRS
    co=c("TILED=YES","COMPRESS=DEFLATE","PREDICTOR=2","BIGTIFF=YES")
    )
file.remove(xml.out)

[1] TRUE

# also remove 'helper' XML if created by saveXML
xml.DC <- paste0(".", strsplit(xml.out,".", fixed = TRUE)[[1]][2], ".DC.xml")
if (file.exists(xml.DC)) file.remove(xml.DC)

[1] TRUE
```

Again, ignore the GDAL warnings.

Check the result by reading into R, summarizing, and plotting. Note this is in the Homolosine projection used by SG250.

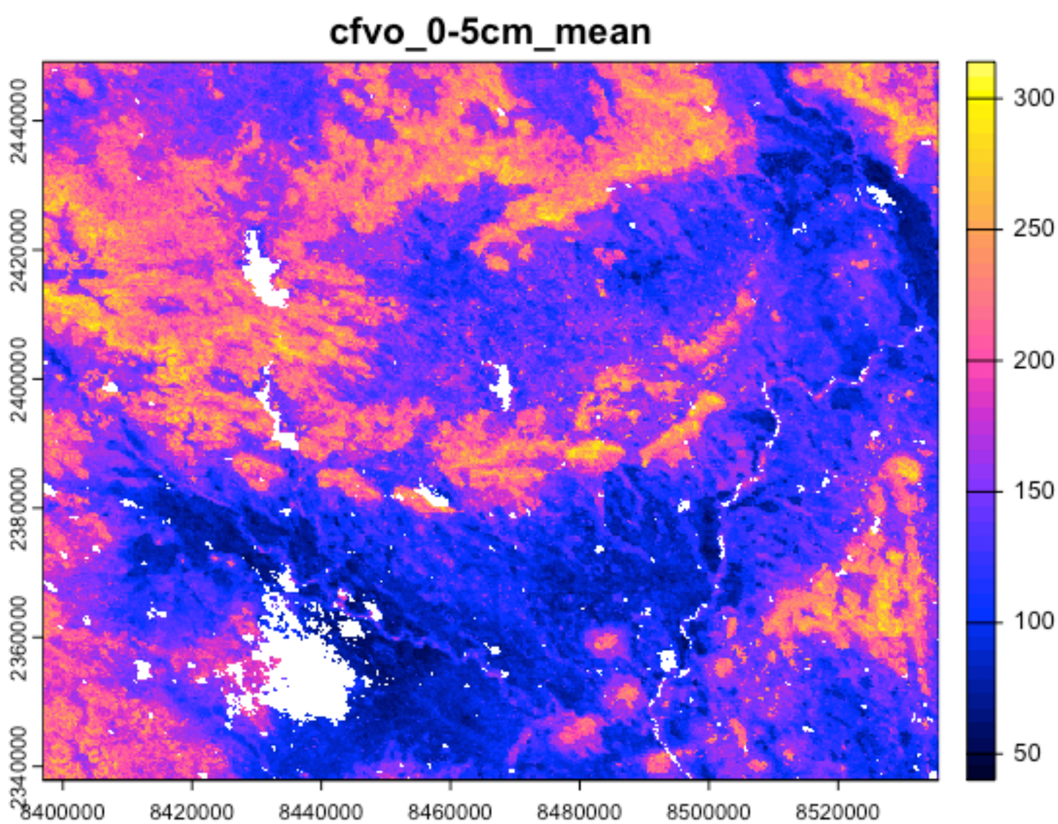
```
r.sg <- terra::rast(file.out)
print(r.sg)

class       : SpatRaster
dimensions  : 445, 555, 1  (nrow, ncol, nlyr)
resolution  : 250, 250  (x, y)
extent      : 8396875, 8535625, 2337875, 2449125  (xmin, xmax, ymin, ymax)
coord. ref. : Interrupted_Goode_Homolosine
source      : cfvo_0-5cm_mean.tif
name        : cfvo_0-5cm_mean

summary(values(r.sg))
```

```
 cfvo_0-5cm_mean
 Min.   : 40.0
 1st Qu.:117.0
 Median :149.0
 Mean   :154.6
 3rd Qu.:194.0
 Max.   :314.0
 NA's   :6980
```

```
terra::plot(r.sg, col=(sp::bpy.colors(50)), main = voi_layer)
```



SoilGrids IGH tile

The 0 values in SoilGrids are in fact un-mapped areas, so convert these to `NA`.

```
quantile(values(r.sg), seq(0,1,by=0.05), na.rm = TRUE)
```

```
  0%   5%  10%  15%  20%  25%  30%  35%  40%  45%  50%  55%  60%  65%  70% 
75% 
  40   80   89   99  109  117  125  131  137  143  149  154  161  169  181 
194 
 80%  85%  90%  95% 100% 
 205  215  226  240  314 

values(r.sg) <- ifelse(values(r.sg) < 1, NA, values(r.sg))
quantile(values(r.sg), seq(0,1,by=0.05), na.rm = TRUE)

  0%   5%  10%  15%  20%  25%  30%  35%  40%  45%  50%  55%  60%  65%  70% 
75% 
  40   80   89   99  109  117  125  131  137  143  149  154  161  169  181 
194 
 80%  85%  90%  95% 100% 
 205  215  226  240  314 

# terra::plot(r.sg, col=(sp::bpy.colors(50)), main = voi_layer)
```

Resample to WGS84 geographic coördinates

Set the CRS and resample to WGS84 geographic coördinates. Use bilinear interpolation – notice that the values are now fractional, rather than integers as in the original SoilGrids layer.

```
st_bbox(r.sg)

   xmin    ymin    xmax    ymax 
8396875 2337875 8535625 2449125 

st_crs(r.sg)$proj4string

[1] "+proj=igh +lon_0=0 +x_0=0 +y_0=0 +datum=WGS84 +units=m +no_defs"

st_crs(r.sg)$epsg       # not in the EPSG database

[1] NA

r.sg.84 <- terra::project(r.sg, "epsg:4326")
quantile(values(r.sg.84), seq(0,1,by=.1), na.rm = TRUE)

       0%       10%       20%       30%       40%       50%       60% 
70% 
 42.51252  90.41739 109.66060 124.78061 137.85013 148.78765 161.05038 
181.04806 
      80%       90%      100% 
204.83351 224.92386 306.69016 

st_crs(r.sg.84)$proj4string

[1] "+proj=longlat +datum=WGS84 +no_defs"
```

```r
st_crs(r.sg.84)$epsg
```

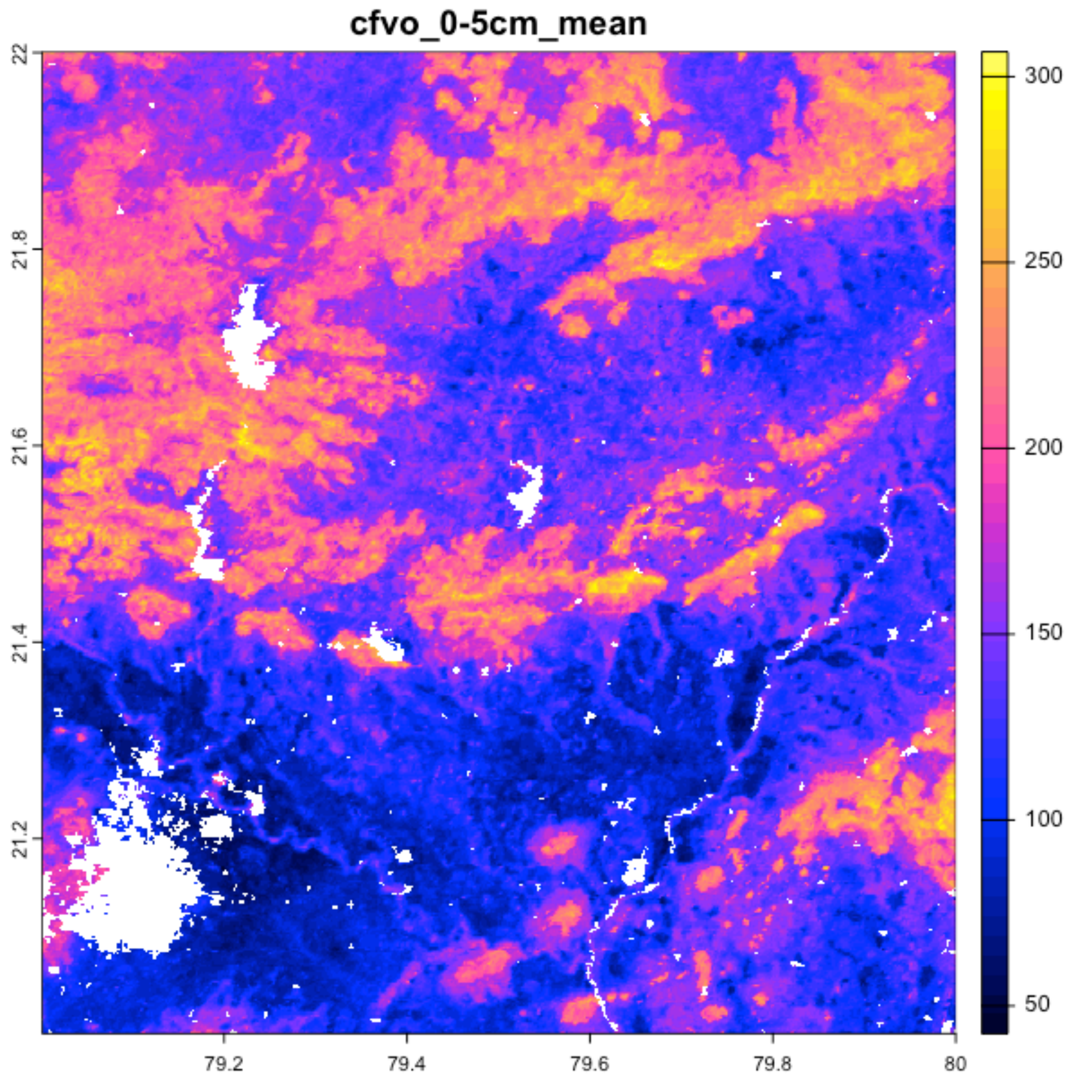
```
[1] 4326
```

Crop this to the long/lat bounding box – recall, the Homolosine bounding box is larger:

```r
r.sg.84.crop <- crop(r.sg.84, bb.ll)
quantile(values(r.sg.84.crop), seq(0,1,by=0.05), na.rm = TRUE)
```

```
       0%        5%       10%       15%       20%       25%       30% 
35% 
 42.51252  80.44707  89.43628  98.25671 107.97195 116.25165 123.78732 
130.95309 
      40%       45%       50%       55%       60%       65%       70% 
75% 
137.24946 142.90861 148.33936 153.96141 160.28127 168.58617 180.75125 
194.81840 
      80%       85%       90%       95%      100% 
206.03329 215.62958 225.40694 237.50547 306.69016 
```

```r
terra::plot(r.sg.84.crop, col=(sp::bpy.colors(50)), main = voi_layer)
```



SoilGrids WGS84 tile

Save tile

Save this map for further processing, e.g., comparing with other DSM products.

Save the tile. Note that the file name includes the property name and depth slice. Specify the float-4 bit datatype and a GeoTIFF "world" file. Each tile is about 750 kB.

```
print(paste("file name:",
      file.out <- paste0(dest.dir.sg, "/", voi_layer, '.tif')))
```

```
[1] "file name: /Users/rossiter/ds_reference/DSM2025/lat2122_lon7980/cfvo_0-
5cm_mean.tif"
```

```r
writeRaster(r.sg.84.crop, file = file.out,
                        overwrite=TRUE, datatype="FLT4S",
                        gdal ="TFW=YES")
```

This can now be imported into analysis scripts.