searching for images with sentinelsat

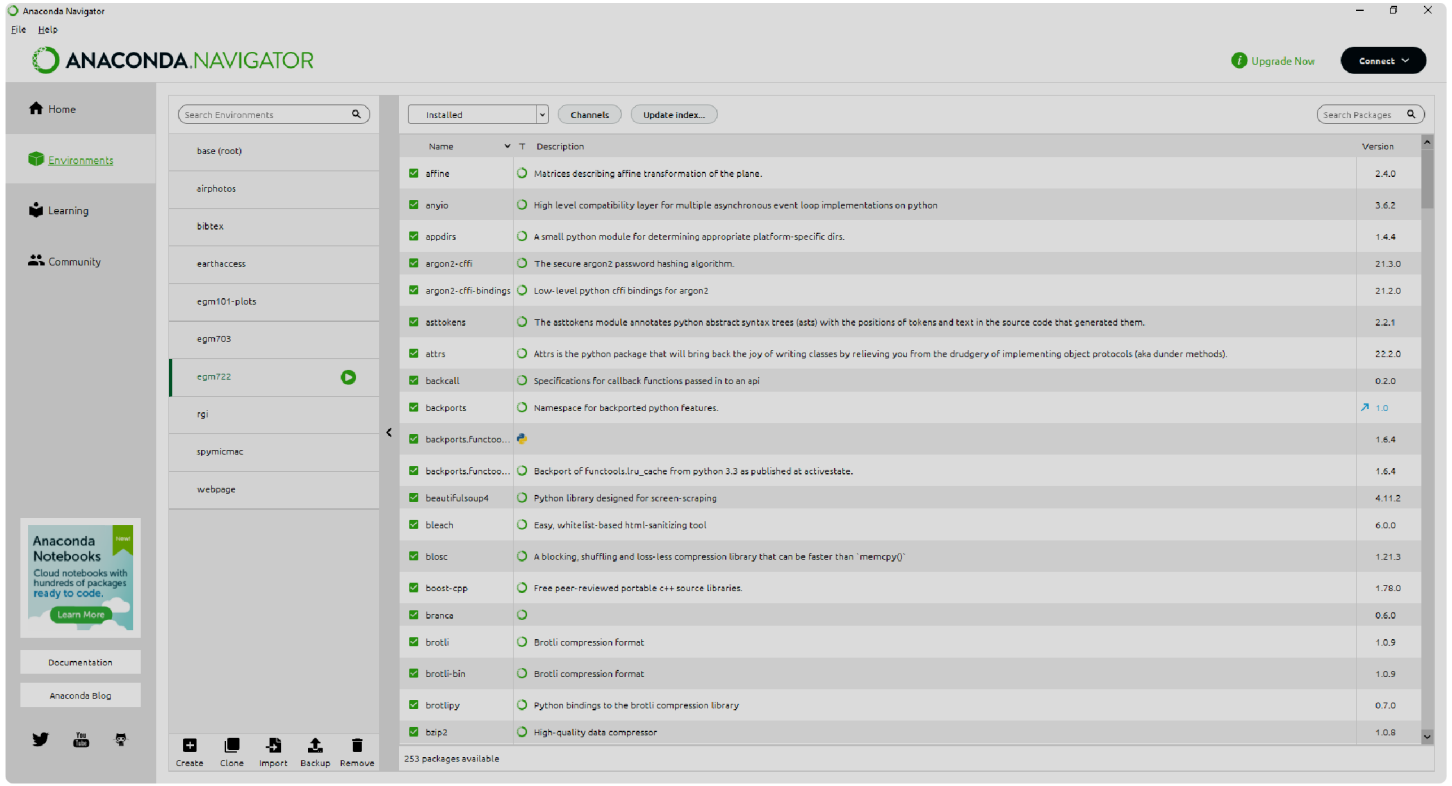
Contents

getting started

The other thing you'll need to do before starting this week's practical is install a new python package, **sentinelsat**, into our **conda** environment.

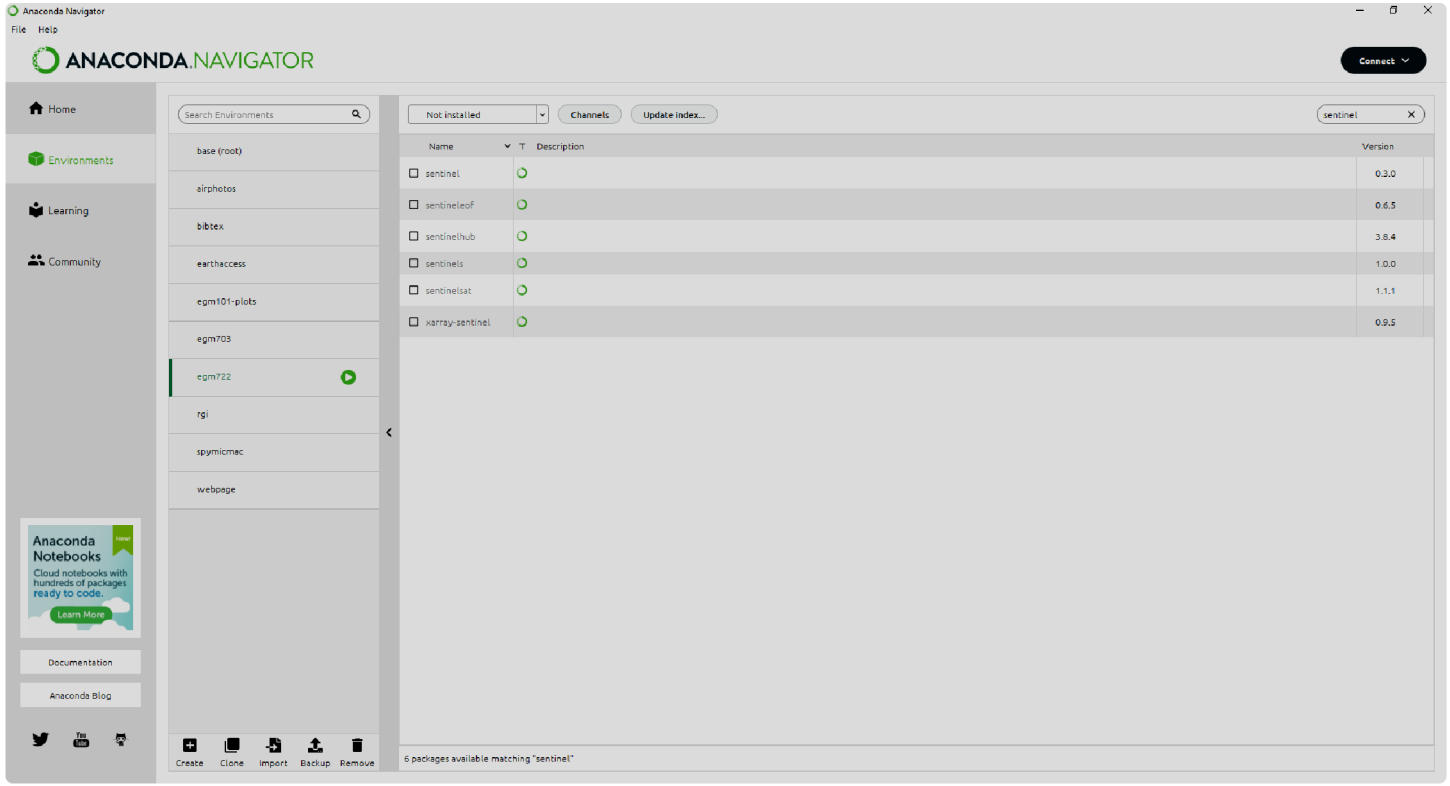
First, open up **Anaconda Navigator**. Make sure to switch to your `egm722` environment, then click on the **Environments** tab:

Skip to main content



Change the list from **Installed** to **Not installed**, then type `sentinel` into the **Search** bar in the upper right-hand corner:

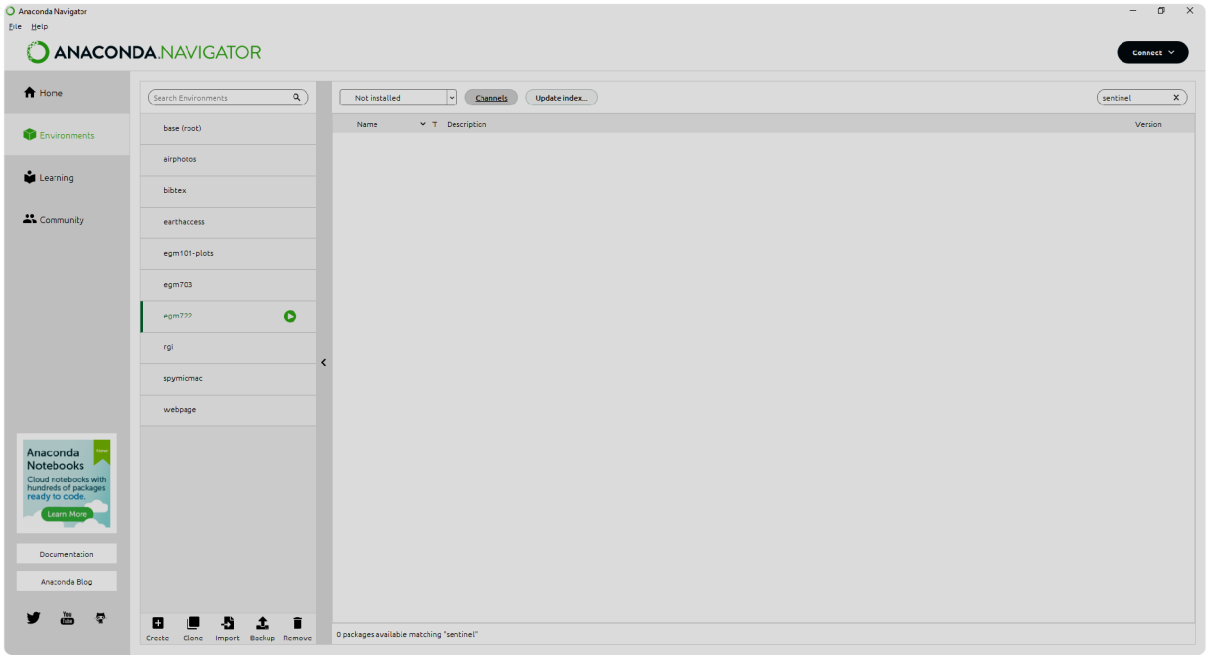
Skip to main content



Skip to main content

Note

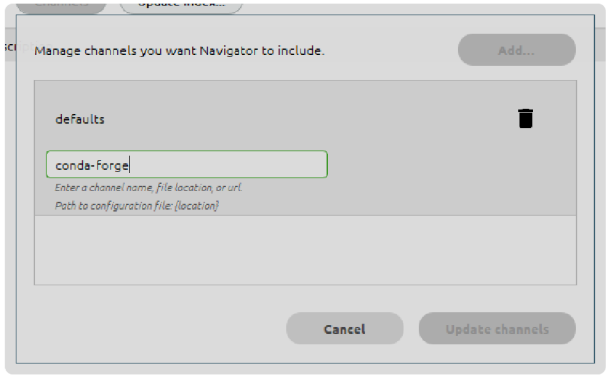
If nothing shows up here, don't panic:



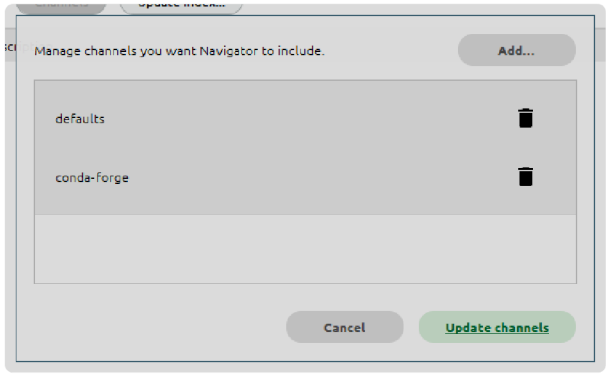
It just means we need to add the `conda-forge` channel to the list of **Channels**.

To do this, first click the **Channels** button, then type `conda-forge`:

Skip to main content



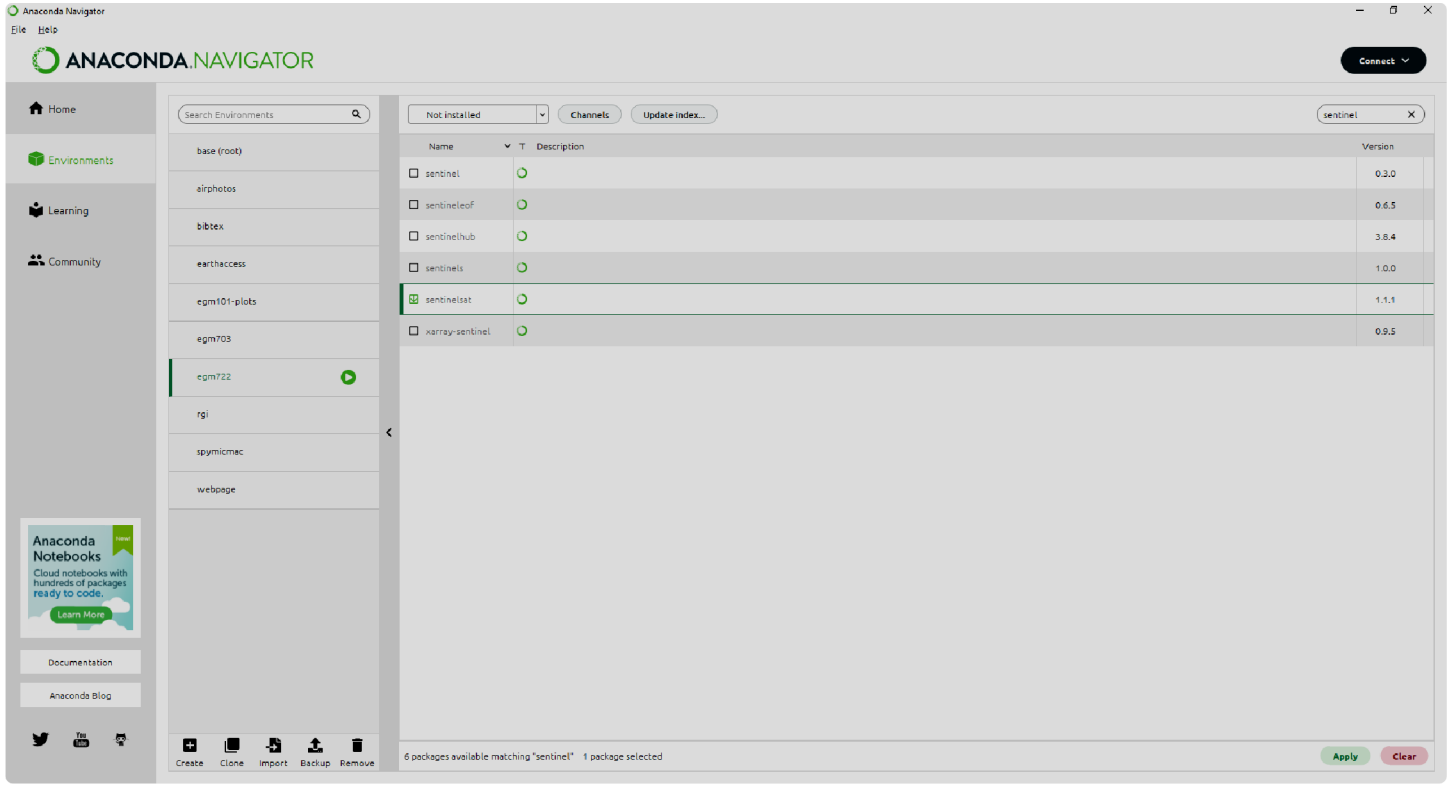
Press **Enter**, and you should see the new channel added to the list:



Click **Update channels**. Once the channel list finishes refreshing, you should see the `sentinelsat` package in the list of available packages.

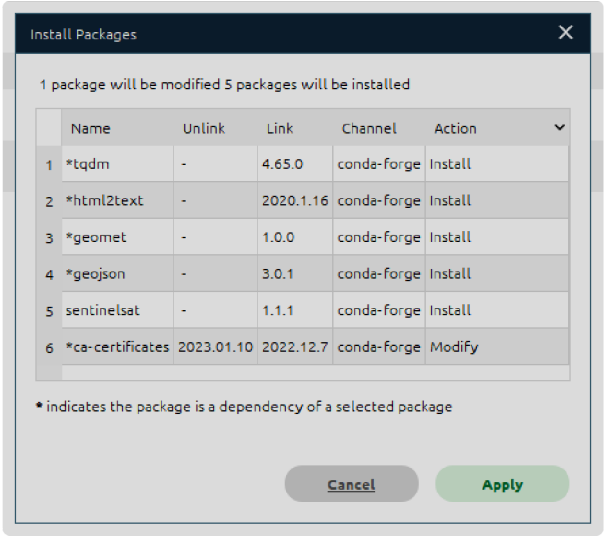
Click the box next to `sentinelsat` to select it, then click **Apply** in the lower right-hand corner:

Skip to main content



You should see the following window open (note that this may take some time):

Skip to main content



Click **Apply**, and the packages will be downloaded/installed in the environment. While you wait for **Anaconda** to finish working, you will be able to move on to the next steps.

Alternatively, go and grab a coffee or tea, and move on to the next steps.

the .netrc file

In order to search for and download data using the sentinelsat API, we will need to authenticate (log in) using your Copernicus Open Access Hub username/password.

The `SentinelAPI` (documentation) class takes your username and password as arguments:

```
connection = SentinelAPI(user, password)
```

Skip to main content

> ⚠ Danger
>
> No, seriously. **NEVER**, **EVER** type your password in plaintext in a script or a jupyter notebook.

Instead, what we'll set up is something called a netrc or `.netrc` file[1], which `SentinelAPI` can read to authenticate you with the Copernicus Open Access Hub.

A netrc file can be used by a number of websites and programs for authentication, making it so that you don't type your password as plaintext in a script or command prompt.

The `.netrc` file is a text file with a simple structure, where each line corresponds to a "machine" or website:

```
machine <website> login <username> password <password>
```

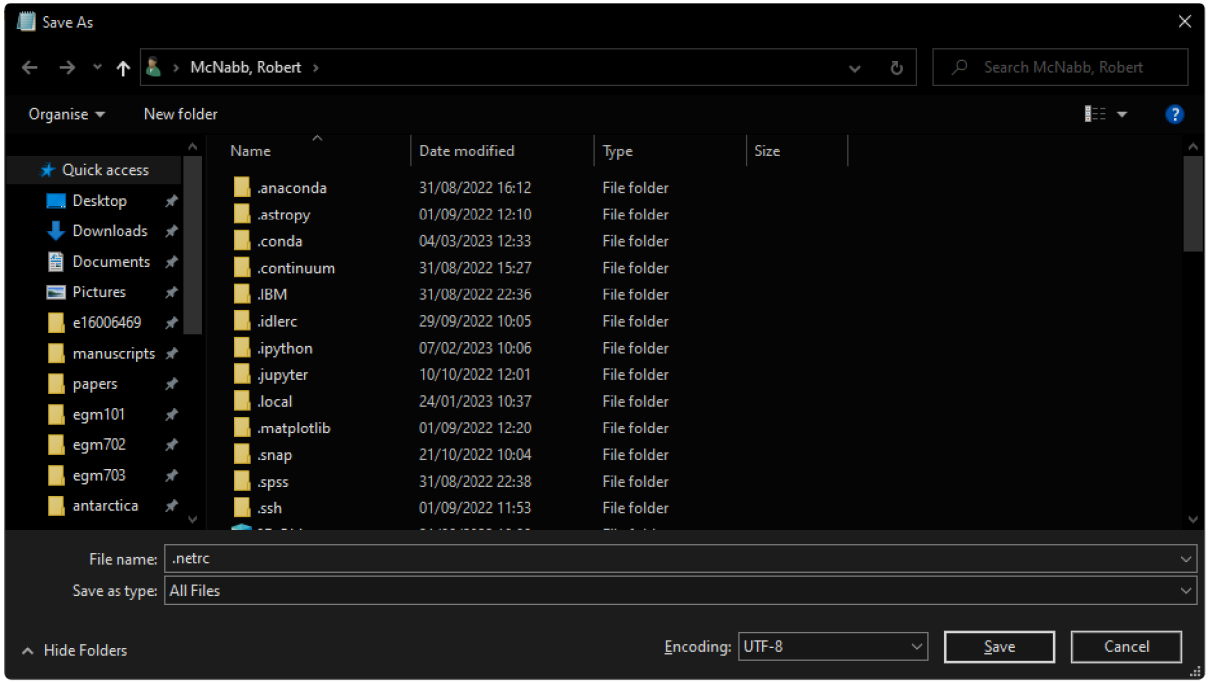
To create the file, open **notepad++** (or **notepad**, or your text editor of choice), and enter the following line:

```
machine apihub.copernicus.eu login <username> password <password>
```

remembering to replace `<username>` with your Copernicus Open Access Hub username, and `<password>` with your Copernicus Open Access Hub password.

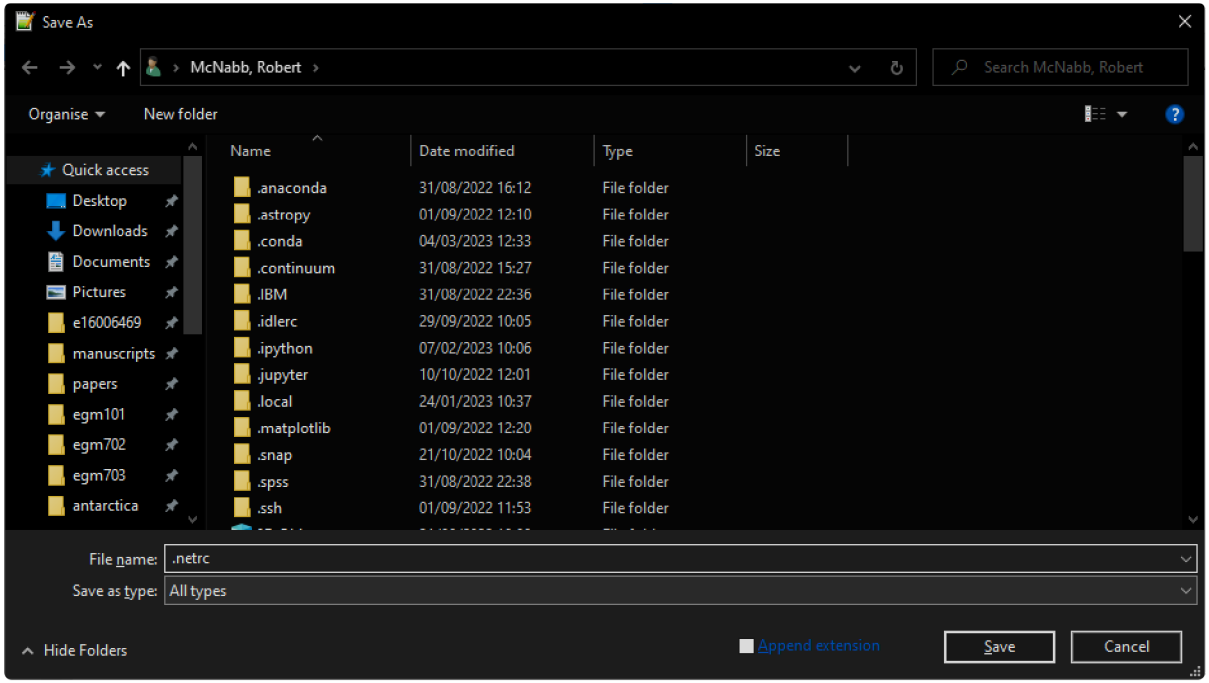
Save the file as `.netrc` to your **Home** directory (on Windows, this should be `C:\Users\<your_username>`). Be sure to select **All Files** for **Save as type**:

Skip to main content



From **notepad++**, you should also uncheck **Append extension**:

Skip to main content



We're not quite done - there's one last thing we'll need to do before moving on.

changing permissions

Note

On MacOS or linux-based systems, enter the following command in a terminal window:

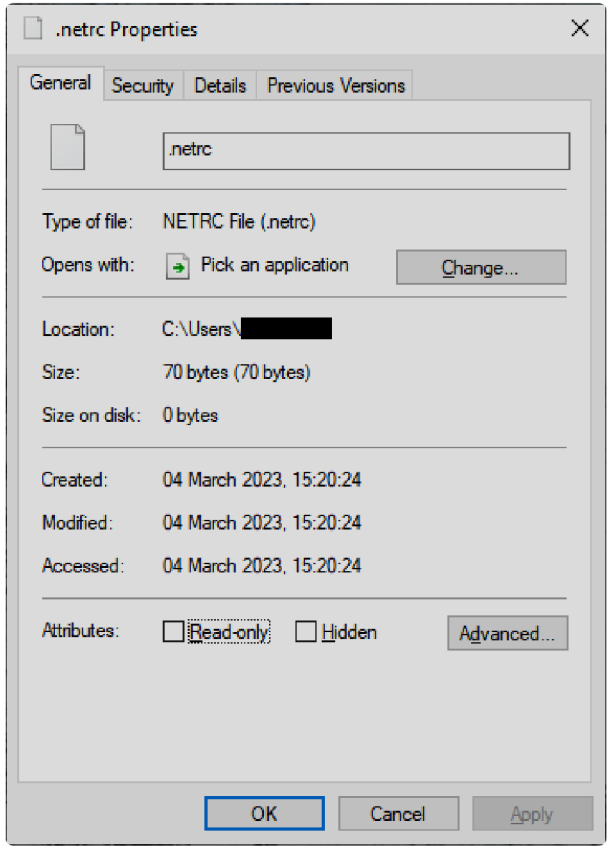
```
chmod 600 ~/.netrc
```

This uses the `chmod` command to change the permissions on the file so that only the *owner* of the file has read/write

Skip to main content

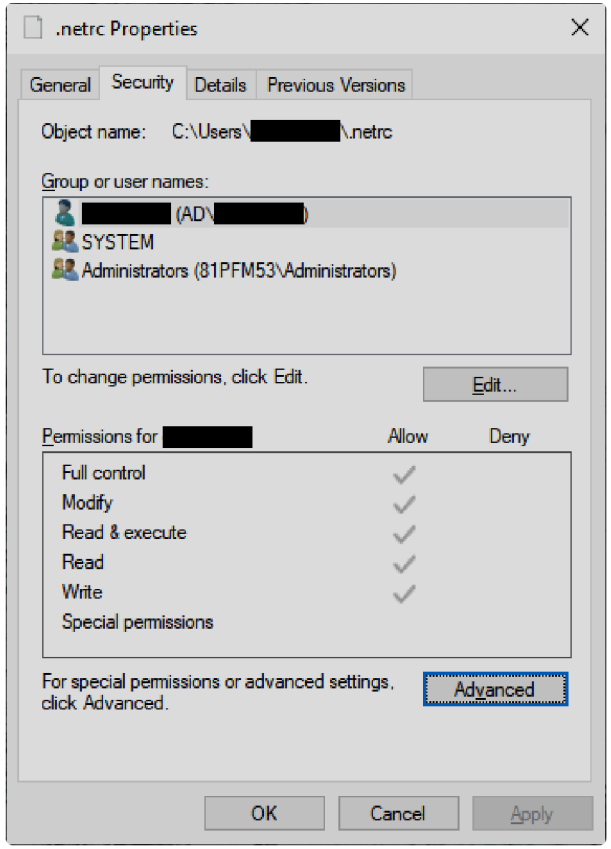
The last step we want to do in setting up the netrc file is to change the *permissions* so that other users can't access the file. To change the permissions of the file in Windows, first open **Windows Explorer**.

From **Windows Explorer**, right-click on the file and select **Properties**:



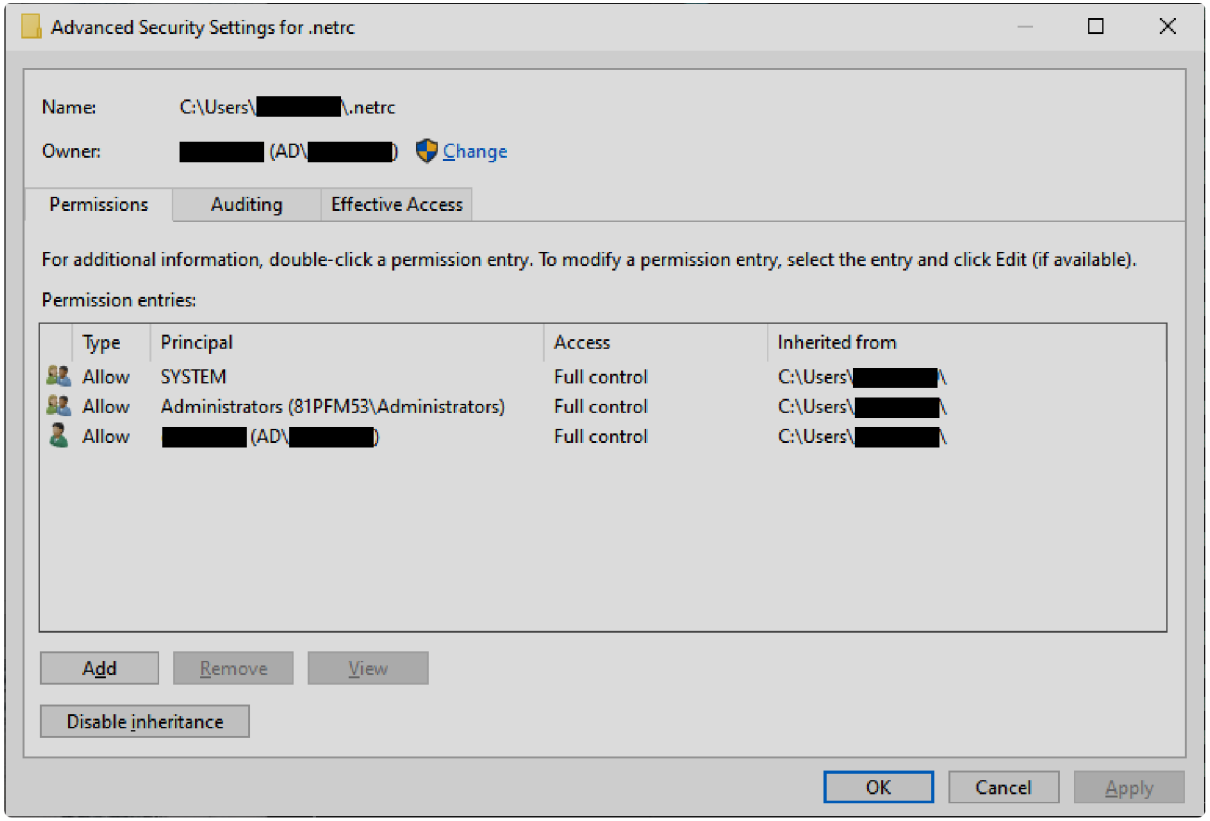
Click on the **Security** tab:

Skip to main content

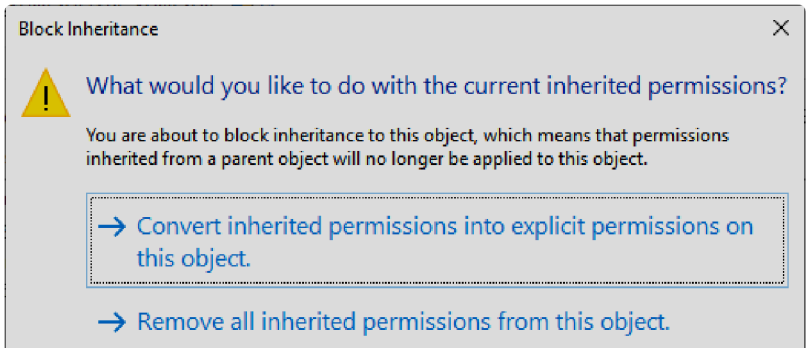


then click **Advanced**:

Skip to main content

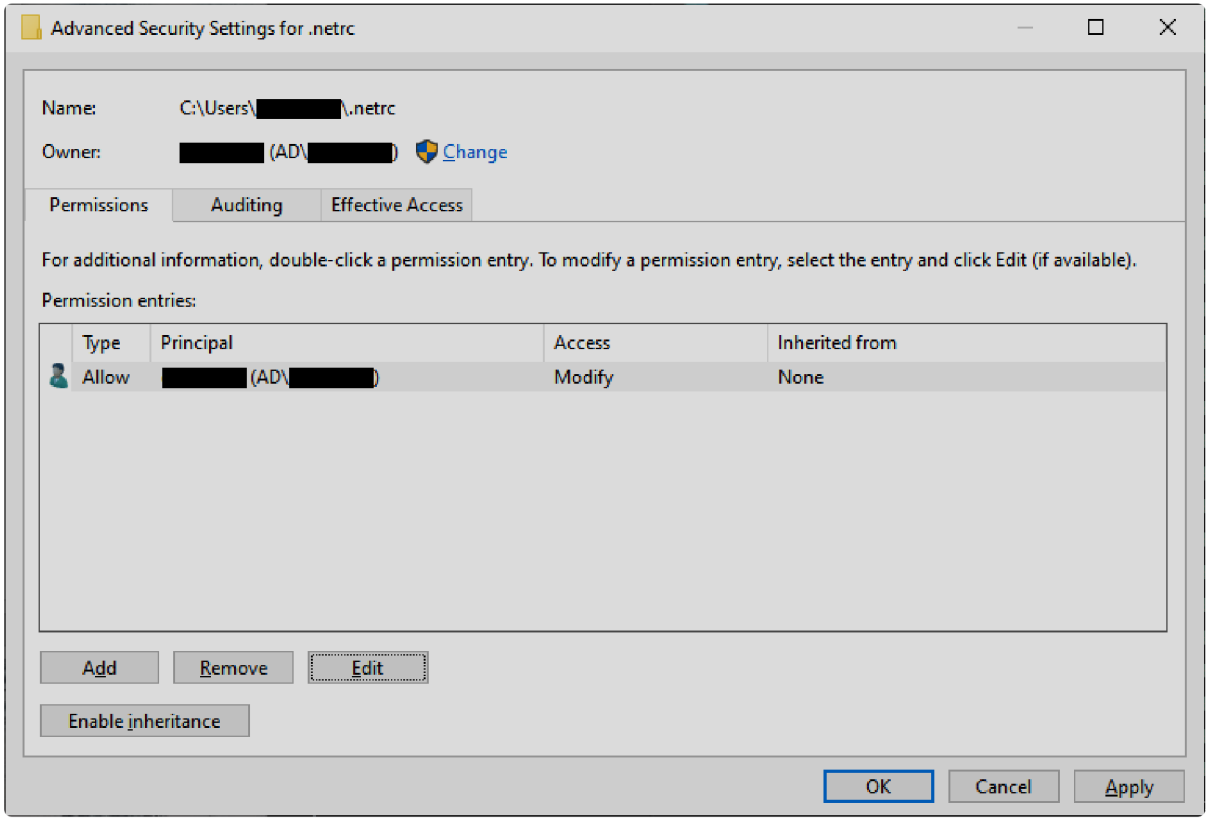


Click **Disable inheritance**, then click **Convert inherited permissions into explicit permissions on this object** in the window that pops up:



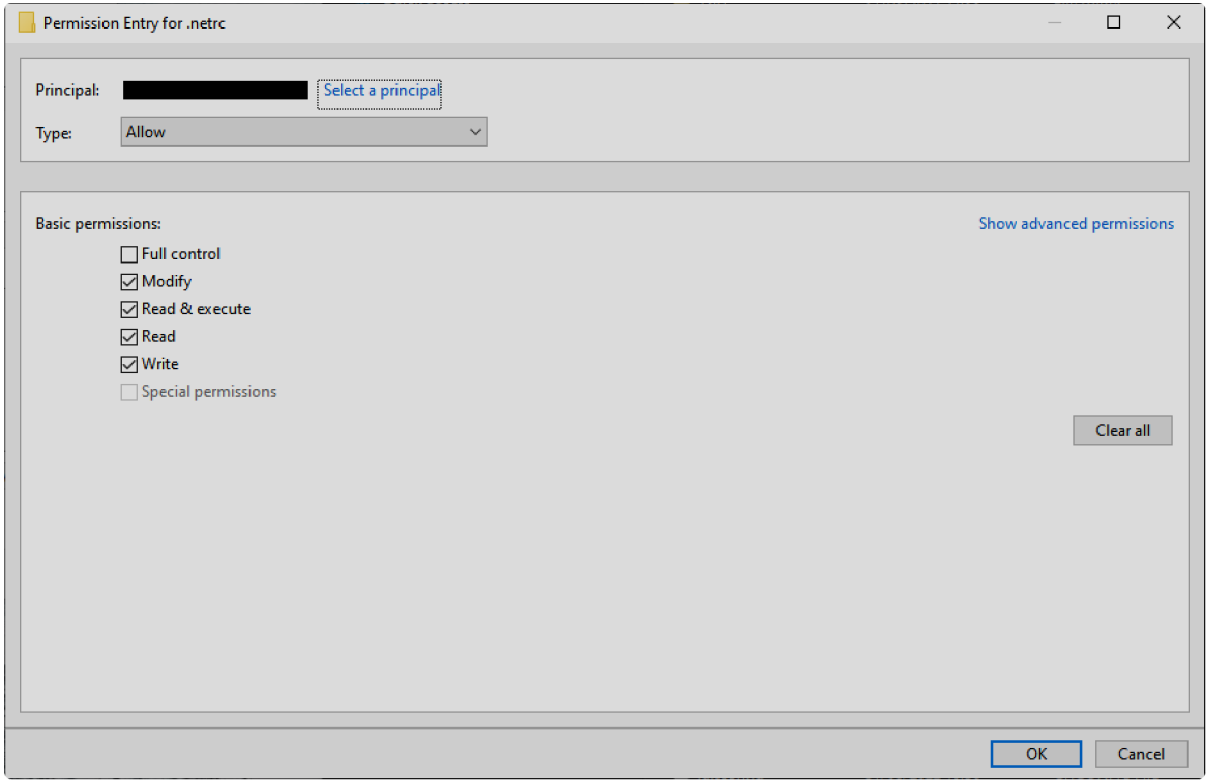
Skip to main content

Now click **Apply**. Next, remove all of the rows from the table that aren't your user name (this should be **SYSTEM** and **Administrators**):



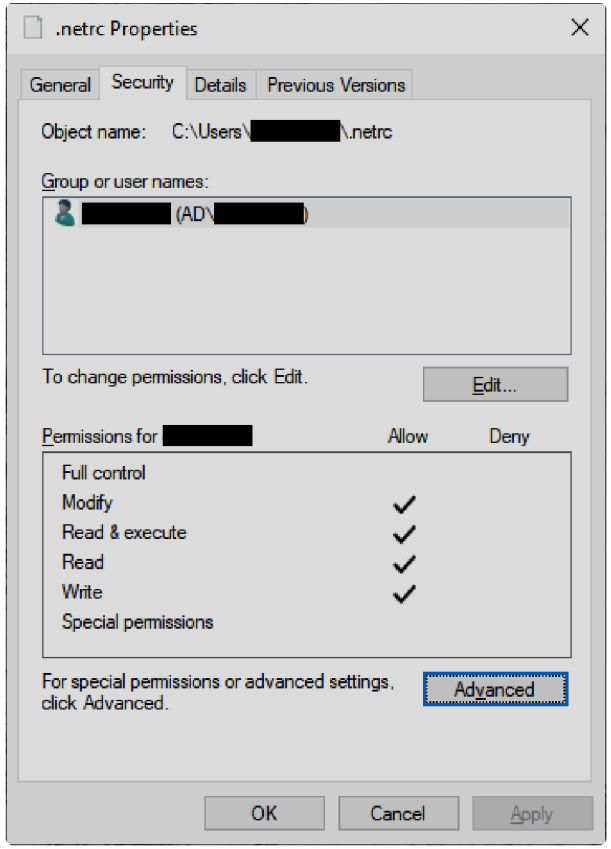
Click **Apply**, then highlight your username and click **Edit**. In the window that opens up, uncheck **Full control**, but make sure that the other 4 available boxes are checked:

Skip to main content



Click **OK**, then **OK** again to close the advanced security settings. You should see the permissions for the file have changed:

Skip to main content



That's it - you should now be able to work through the notebook (assuming that you have correctly entered your credentials in the `.netrc` file, that is).

At this point, you can launch Jupyter Notebooks from the command prompt, or from Anaconda Navigator, and begin to work through the exercise.

> **Note**
>
> Below this point is the **non-interactive** text of the notebook. To actually run the notebook, you'll need to follow the

Skip to main content

Gena Rowlands

overview

Up to now, you have gained some experience working with basic features of python, used cartopy and matplotlib to create a map, explored using shapely and geopandas to work with vector data, and explored using rasterio and numpy to work with raster data.

In this example, we'll see how we can use an application programming interface (API) to query and download Sentinel data, using the SentinelSat API. As part of this, we'll also introduce a few more geometric operations using `shapely` that you may find useful.

objectives

In this example, you will:

- Use `shapely` to get the *unary union* of a collection of shapes
- Use `shapely` to find the minimum bounding rectangle of a geometry
- Use the SentinelAPI to search for Sentinel-2 images
- Calculate the fractional overlap between shapes
- Use the SentinelAPI to download images

Skip to main content

data provided

In this example, we will be using the `Counties` shapefile that we used in Week 2.

getting started

To get started, run the following cell to import the packages that we'll use in the practical.

```
import os
import geopandas as gpd
from sentinelsat import SentinelAPI, make_path_filter
from IPython import display # lets us display images that we download
import shapely
```

preparing a search area

Before we get to using the API to search for images, we'll see how we can use existing data, like the `Counties` shapefile we used in Week 2, to help us search for images.

We won't be able to use particularly complicated shapes, but we can use a combination of GIS/geometric operations to get a simple outline of our data, which can be used for the search.

First, we'll load the data using `geopandas`, making sure to transform from the original CRS to WGS84 latitude/longitude (`epsg=4326`):

```
counties = gpd.read_file('../Week2/data_files/Counties.shp').to_crs(epsg=4326)
```

Skip to main content

Next, we'll use the `geopandas.Series.unary_union` attribute (documentation) to get a combination of all of the County outlines in a single geometry feature:

```
# gets a single polygon (or multipolygon) composed of the individual polygons
outline = counties['geometry'].unary_union

outline # note that in a jupyter notebook, this actually displays the polygon.
```

In the output of the cell above, we can see that the `outline` shape is indeed the combination of all of the individual county outlines.

We could use this as an input to our search, but we'll look at one additional attribute of a `shapely` **Polygon** that we can use to get a bounding box of the geometry - the `minimum_rotated_rectangle` (documentation):

```
# gets the minimum rotated rectangle that covers the outline
search_area = outline.minimum_rotated_rectangle

search_area
```

You can see above that this gives a boundary box of the polygon, but rather than being a simple rectangle made of the maximum/minimum coordinates, it's rotated to be as small as possible while still covering the entire geometry.

This way, we minimize the area outside of the area of interest (Northern Ireland) within our search area, while still making sure to cover the entire area of interest.

Finally, if we look at the docstring for `SentinelAPI.query()` (documentation), we see that the `area` argument needs to be a `str`:

Skip to main content

Specifically, it needs to be a "Well-Known Text (WKT)" representation of the geometry.

For a `shapely` geometry, the WKT representation of the geometry is stored in the `wkt` attribute:

```
# displays the search area wkt
print(search_area.wkt)
```

That's all we need to be able to search for images that intersect with a given geometry. Once we have this, we can connect to the API and start the query.

searching the archive for images

connecting to the api

To connect to the API, we first create a **SentinelAPI** object (documentation):

```
api = SentinelAPI(None, None) # remember - don't type your username and password into a jupyter notebook!
```

From the API reference for `sentinelsat`, we can see that we either type in the username and password as a string (**a terrible idea - don't do this!**), or we use `None` to use the `.netrc` file that we created earlier.

If there are no error messages or warnings, the connection was successfully created, and we can move on to searching for images.

Skip to main content

searching for images

As we saw earlier, the method we'll use is `api.query()`.

For this example, we'll use the following arguments for the search:

- `area`: the search area to use
- `date`: the date range to use. We'll look for all images from February 2023.
- `platformname`: we're going to limit our search to Sentinel-2, but note that there are other options available
- `producttype`: we'll search for the Sentinel-2 MSI Level 2A (surface reflectance) products
- `cloudcoverpercentage`: we want (mostly) cloud-free images, so we'll search for images with < 30% cloud cover

To see what additional arguments are available, you can check the SentinelAPI API reference, or the Open Access Hub API reference for additional keywords to use.

```
products = api.query(search_area.wkt, # use the WKT representation of our search area
                     date=('20230201', '20230228'), # all images from February 2023
                     platformname='Sentinel-2', # the platform name is Sentinel-2
                     producttype='S2MSI2A', # surface reflectance product (L2A)
                     cloudcoverpercentage=(0, 30)) # limit to 10% cloud cover
```

The output of `api.query()` is a **dict** object, with the product name the `key` and the `value` being the metadata.

To see how many images were returned by the search, we can check the length of the **dict** object, which tells us the number of `item`s (`key`/`value` pairs) in the **dict**:

```
nresults = len(products)
print(f'Found {nresults} results')
```

Skip to main content

You should hopefully see that the search has returned 11 results.

To look at the first one returned, we can use the built-ins `next()` (documentation) and `iter()` (documentation), which returns the first item that was entered into the **dict**:

```
result = next(iter(products)) # get the "first" item from the dict
products[result] # show the metadata for the first item
```

And, we can also download the browse image for this product, using `SentinelAPI.download_quicklook()` (documentation):

```
qlook = api.download_quicklook(result) # download the quicklook image for the first result
display.Image(qlook['path']) # display the image
```

In this example, we might notice a small problem - while this image technically does intersect our area of interest, it does so only barely. Northern Ireland is the small bit of land in the lower left-hand corner of this image - most of the image is of Scotland and the Irish Sea.

In the next section, we'll see one way that we can make sure that we're only getting images that mostly intersect with our area of interest.

filtering by overlap

To start, we use `SentinelAPI.to_geodataframe()` (documentation) to convert the results into a **GeoDataFrame**:

```
product_geo = SentinelAPI.to_geodataframe(products) # convert the search results to a geodataframe
product_geo.head() # show the first 5 rows of the geodataframe
```

Skip to main content

Now, we can iterate over **GeoDataFrame** to calculate the intersection of the image footprint with the outline of Northern Ireland:

```
for ind, row in product_geo.iterrows():
    intersection = outline.intersection(row['geometry']) # find the intersection of the two polygons
    product_geo.loc[ind, 'overlap'] = intersection.area / outline.area # get the fractional overlap

print(product_geo.overlap) # show the fractional overlap for each index
```

In this example, the third image, `80558644-2e31-48b9-acd5-5d1475dfc1bf` has 43% overlap with the outline of Northern Ireland; none of the other images have more than 20%.

Rather than copying this down, we can use `geopandas.GeoSeries.argmax()` (documentation) to find the integer location of the largest value in the `overlap` column:

```
max_index = product_geo.overlap.argmax() # get the integer location of the largest overlap value
print(max_index) # should be 2
```

Then, we get the **GeoDataFrame** index that corresponds to that integer location:

```
best_overlap = product_geo.index[max_index] # get the actual index (image name) with the largest overlap
print(product_geo.loc[best_overlap]) # show the metadata for the image with the largest overlap
```

With this, we can use `api.download_quicklook()` (documentation) to download the quicklook image for the result that has the largest overlap with the outline of Northern Ireland:

```
qlook = api.download_quicklook(best_overlap) # download the quicklook image for the first result
```

Skip to main content

So that's a little bit better - at least with this image, we can see much more of Northern Ireland (and the ever-present clouds).

That's all for right now - the next few cells provide examples for how you can download the actual image data.

downloading images

Remember that these are very large files (each granule is ~1GB), so you should only run these cells if you actually want to download the data!

downloading an individual image

We can use `SentinelAPI.download()` (documentation) to download a single product, given the Product ID:

```
api.download(best_overlap) # downloads the first result
```

download an individual image, but only the image bands

This example uses the `nodefilter` argument along with `make_path_filter()` (documentation) to only download the image bands (files that match the format `*_B*.jp2`):

```
api.download(first_result, # downloads the first result
             nodefilter=make_path_filter("*_B*.jp2")) # only download the image bands (optional)
```

Skip to main content

download all images from a list of products

Finally, `SentinelAPI.download_all()` (documentation) will download all of the products in a list.

Again, these are very large files, so you should only run the following cell if you actually want to download all of the images returned by the API!

```
api.download_all(products,
                 n_concurrent_dl=5, # allow up to 5 concurrent downloads
                 nodefilter=make_path_filter("*_B*.jp2")) # only down the image bands (optional)
```

notes and references

[1] on Windows, some programs may look for a `_netrc` file in your **Home** directory, rather than `.netrc`.

Previous
raster data using rasterio

Next
zonal statistics using rasterstats