Link:

<https://planetarycomputer.microsoft.com/dataset/3dep-lidar-dsm>

USGS 3DEP Lidar Digital Surface Model

**Overview summary**

This collection is derived from the USGS 3DEP COPC collection. It creates a Digital Surface Model (DSM) using pdal.filters.range to output a collection of Cloud Optimized GeoTIFFs, removing all points that have been classified as noise.

STAC Collection

<https://planetarycomputer.microsoft.com/api/stac/v1/collections/3dep-lidar-dsm>

Providers

Landrush(processor, producer)

USGS (processor, producer, licensor)

Microsoft (host, processor)

License

About 3DEP Products & Services

Temporal Extent

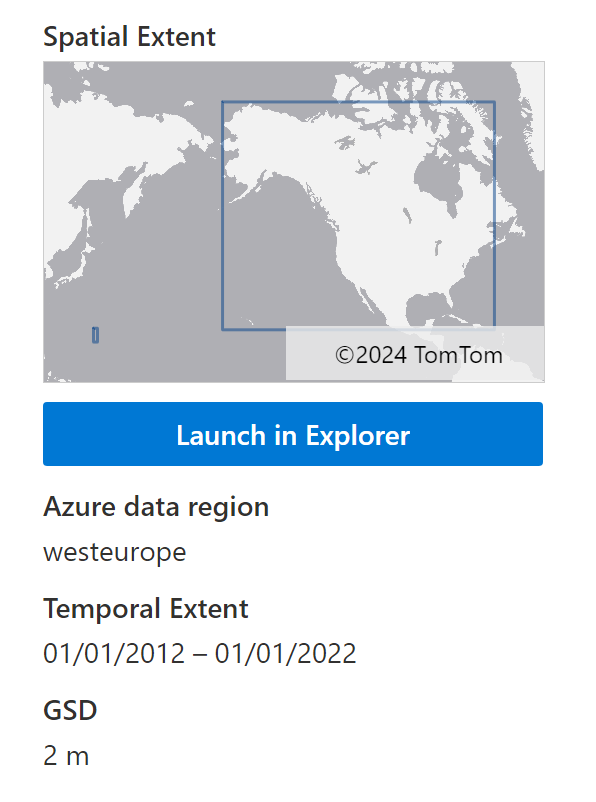
01/01/2012 – 01/01/2022

GSD

**2 m**

Item-level Assets

Dataset items contain the following assets.



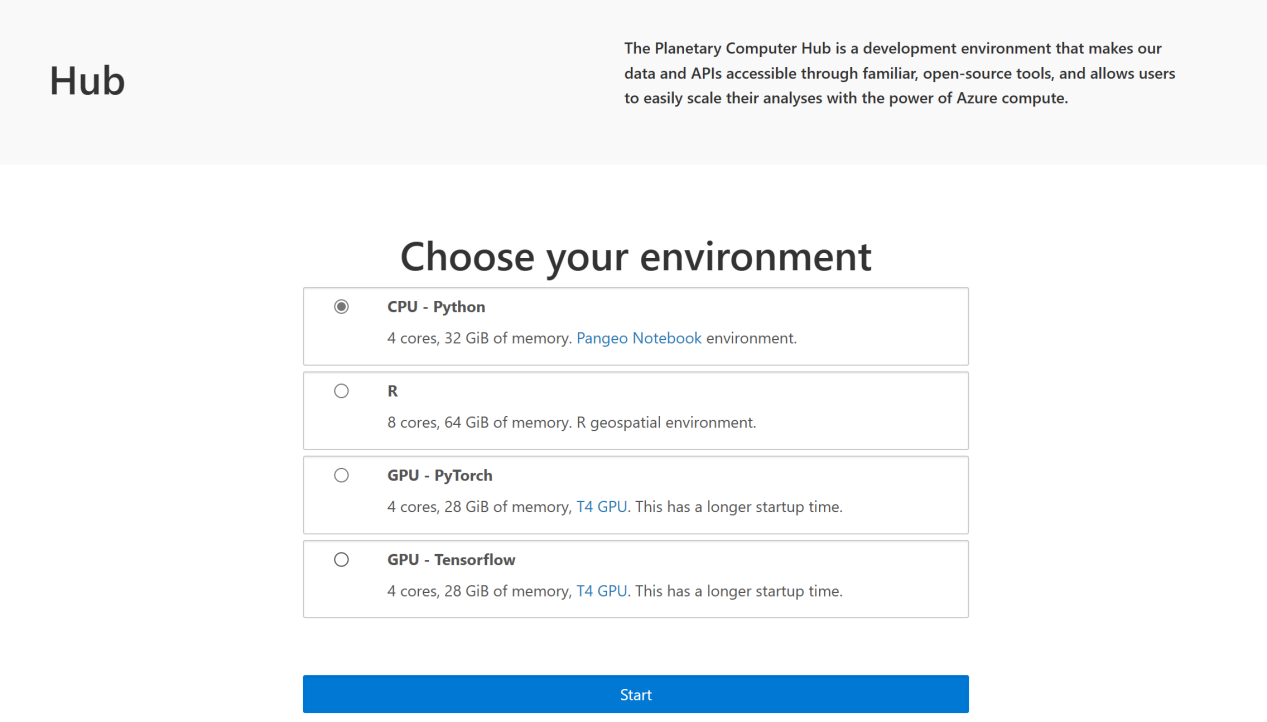
To use the Planetary Computer STAC **API** to access 3DEP lidar COG data, you must first register an account. After registering the account, you can obtain the API. The specific registration process can be found at

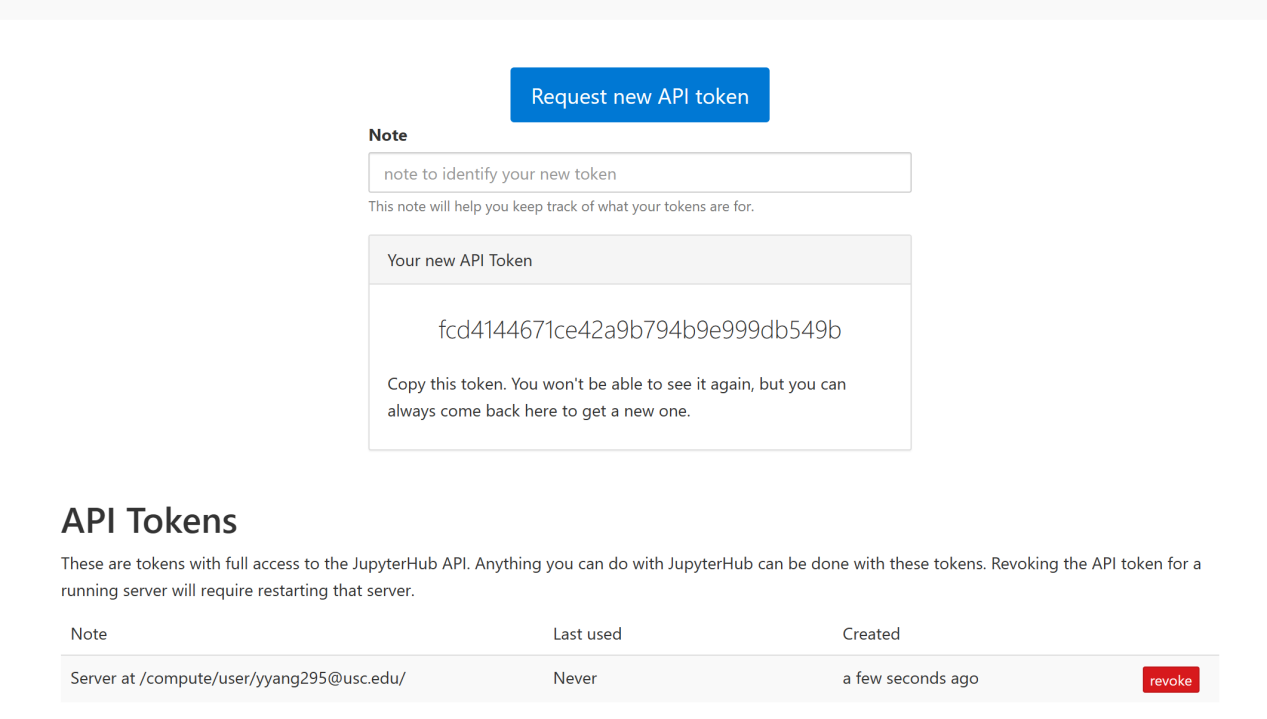
<https://planetarycomputer.microsoft.com/account/request>

The Planetary Computer contains a set of data sets derived from the USGS 3DEP lidar program. Raw data are provided as a collection of COPC assets. In addition, various derivatives such as **Intensity and Height Above Ground** are available.

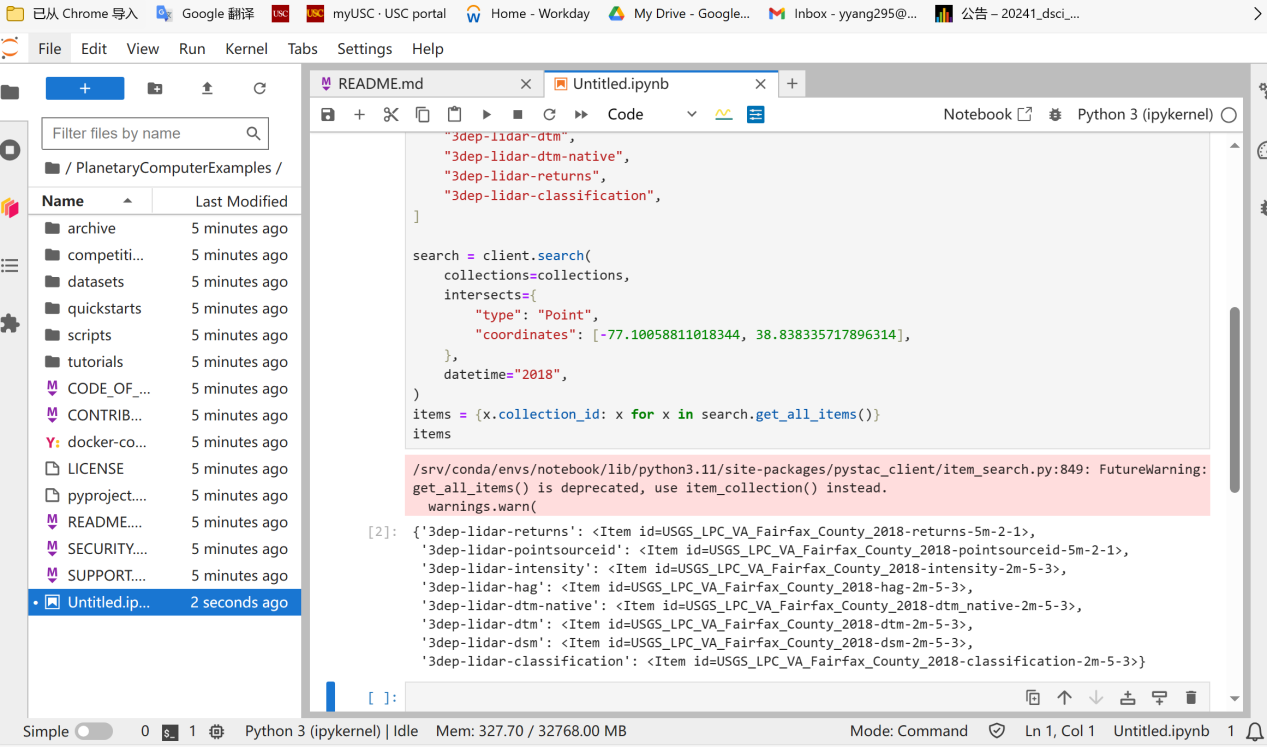
Environment settings

This notebook can be used with or without an API key, but with an API key you will get looser data access. Planetary Computer Hub is pre-configured to use your API key. You can view more details after entering the Hub





This notebook can be used with or without an API key, but with an API key you will get looser data access. Planetary Computer Hub is pre-configured to use your API key. At the same time, you can use the python notebook environment provided by Hub, which is based on CPU and GPU. I think CPU-based is enough.



Query a set of geospatial datasets named `collections` using an object named `client` (the client of the Geospatial Data API).

1. `collections` is a list containing 8 strings, each string represents a different geospatial data set, such as "3dep-lidar-hag", "3dep-lidar-dsm", etc.

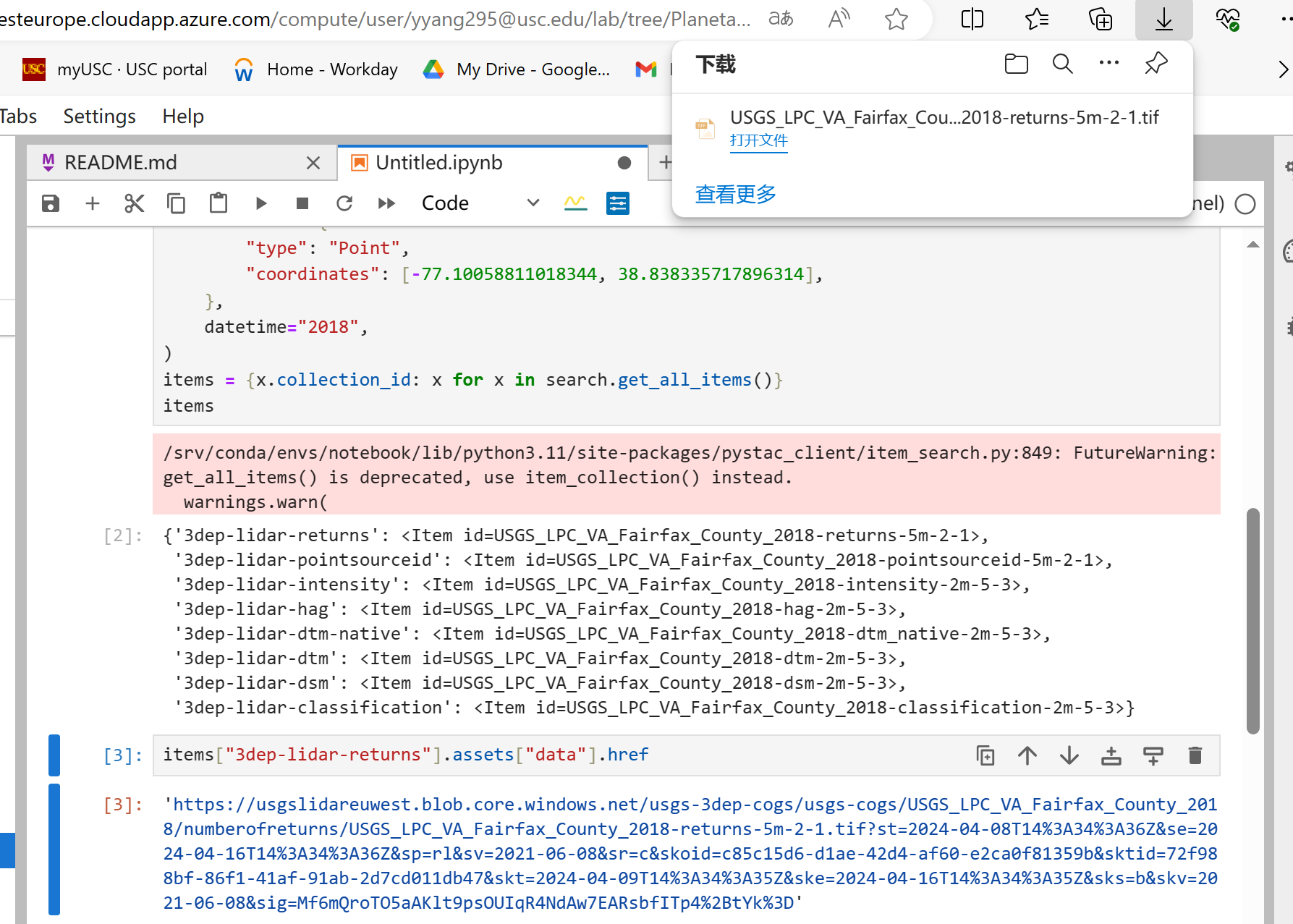
2. The `client.search()` method is used to query these data sets. Query conditions include:

- `collections`: List of data sets to be queried.

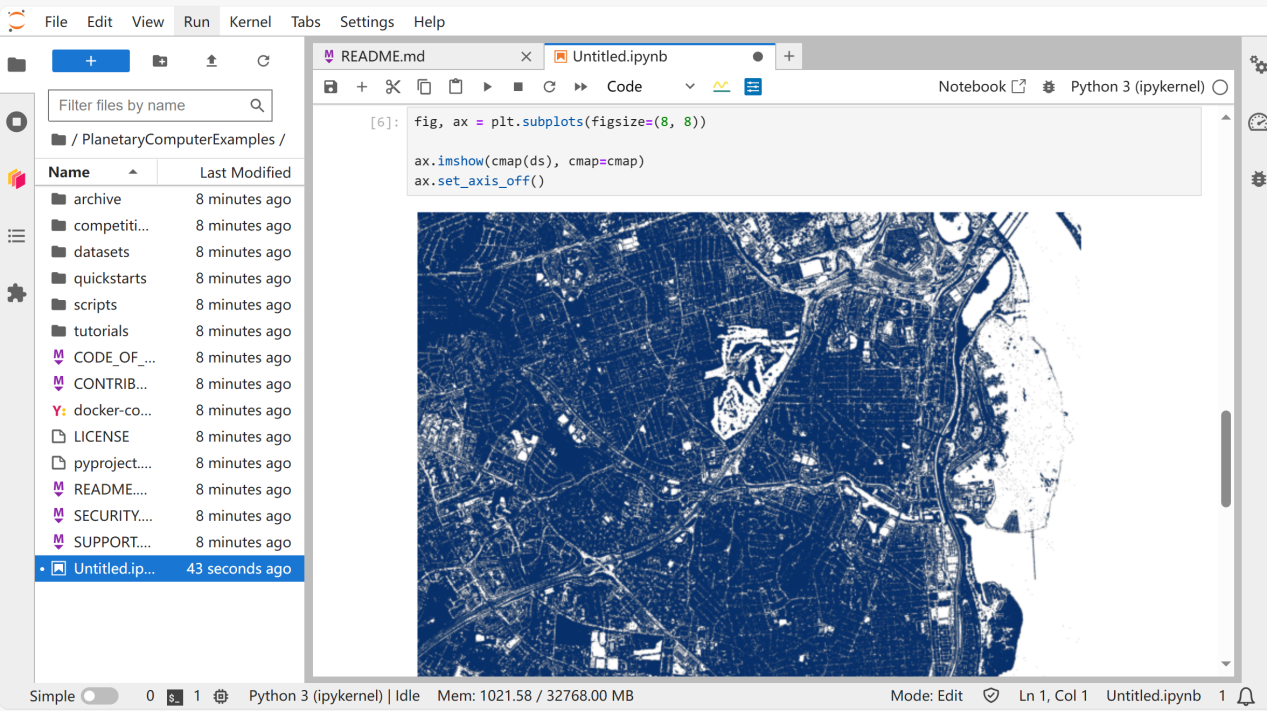
- `intersects`: A GeoJSON object containing coordinates representing the geographical area to be queried. The coordinates here are (-77.10058811018344, 38.838335717896314), which is a latitude and longitude point (in our project, we can use los angeles boyle heights)

3. `search.get\_all\_items()` returns all items in the query results.

4. The last line `items = {x.collection\_id: x for x in search.get\_all\_items()}` is a dictionary comprehension, which will use `x.collection\_id` as the key for each item `x` in the query result, ` x` as a value, constructed into a dictionary `items`. In this way, we can quickly obtain the query result items corresponding to the data set through the ID of the data set.



You will get a download link later to download these data.



**Height Above Ground**

This COG type is generated using the Z dimension of the COPC data data and removes noise, water, and using pdal.filters.smrf followed by pdal.filters.hag\_nn.

This code defines a piecewise linear color map (LinearSegmentedColormap) for mapping values to colors. ,

1. `pairs` is a list containing tuple pairs, each tuple pair consists of an interval and an RGBA color value. The first tuple `((-900, 1), (0, 0, 0, 0))` means that values less than or equal to 1 are mapped to black (0,0,0,0) with an opacity of 0, which is complete transparent.

2. `zip(\*pairs)` Unzips pairs into two lists, containing all intervals and color values respectively.

3. `intervals, colors = zip(\*pairs)` Assign the two decompressed lists to `intervals` and `colors` respectively.

4. `nodes = np.array([x[1] for x in intervals]).astype(float)` Extract the upper limit of all intervals from intervals and construct a numpy array.

5. `nodes -= np.abs(nodes.min())` Subtract the minimum value (possibly a negative number) in nodes to ensure that the minimum value is 0.

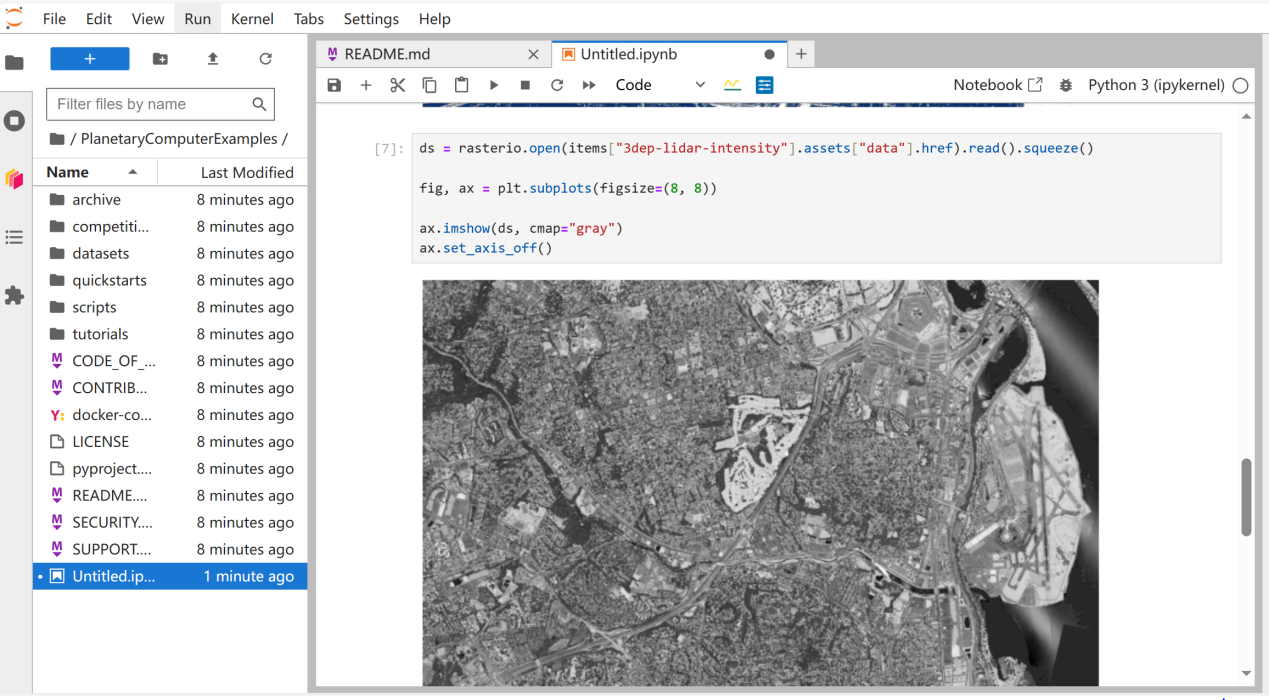
6. `nodes /= nodes.max()` normalizes nodes to the range of 0 to 1.

7. `colors = [np.asarray(c)/255 for c in colors]` Divide each RGBA tuple in colors by 255 to convert it to a floating point range from 0 to 1.

8. `cmap = matplotlib.colors.LinearSegmentedColormap.from\_list("hag", list(zip(nodes, colors)))` Use nodes as color segmentation points and colors as corresponding RGBA values to create a file named "hag" LinearSegmentedColormap object cmap.

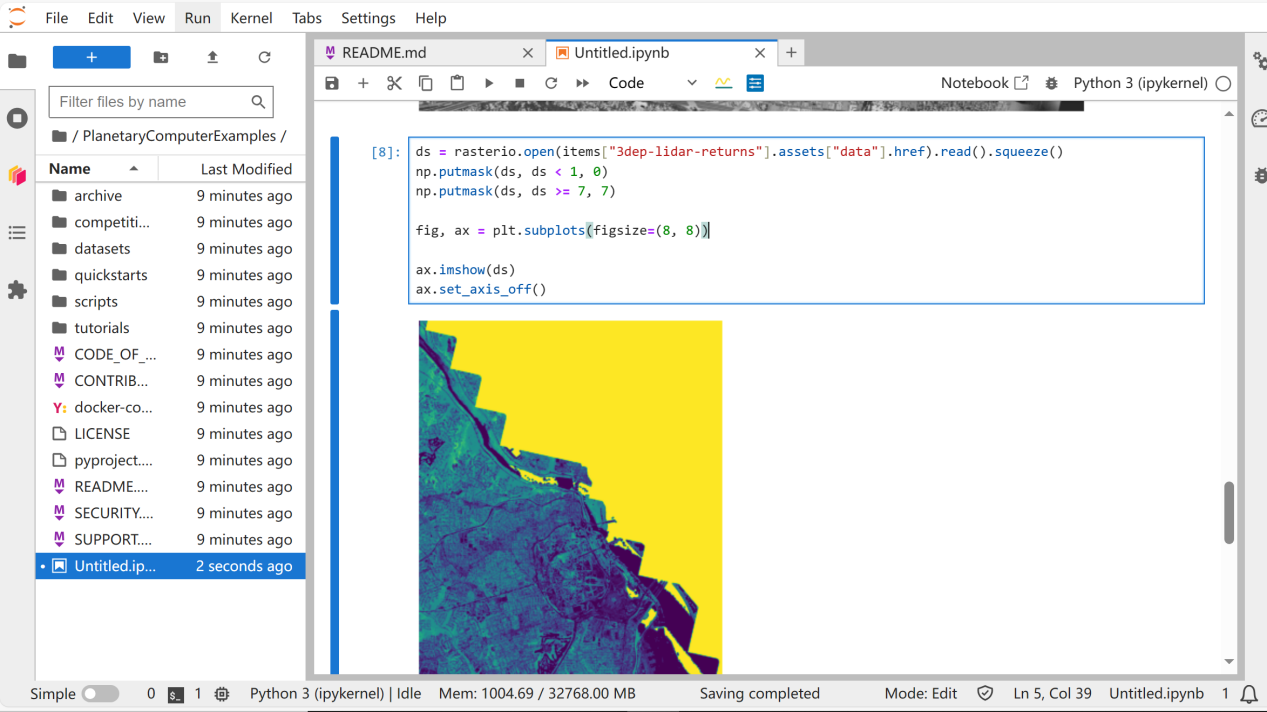
### Intensity

This collection is derived from the [USGS 3DEP COPC collection](https://planetarycomputer.microsoft.com/dataset/3dep-lidar-copc). It is a collection of Cloud Optimized GeoTIFFs representing the pulse return magnitude.



**Returns**

This collection is derived from the USGS 3DEP COPC collection. It is a collection of Cloud Optimized GeoTIFFs representing the number of returns for a given pulse.



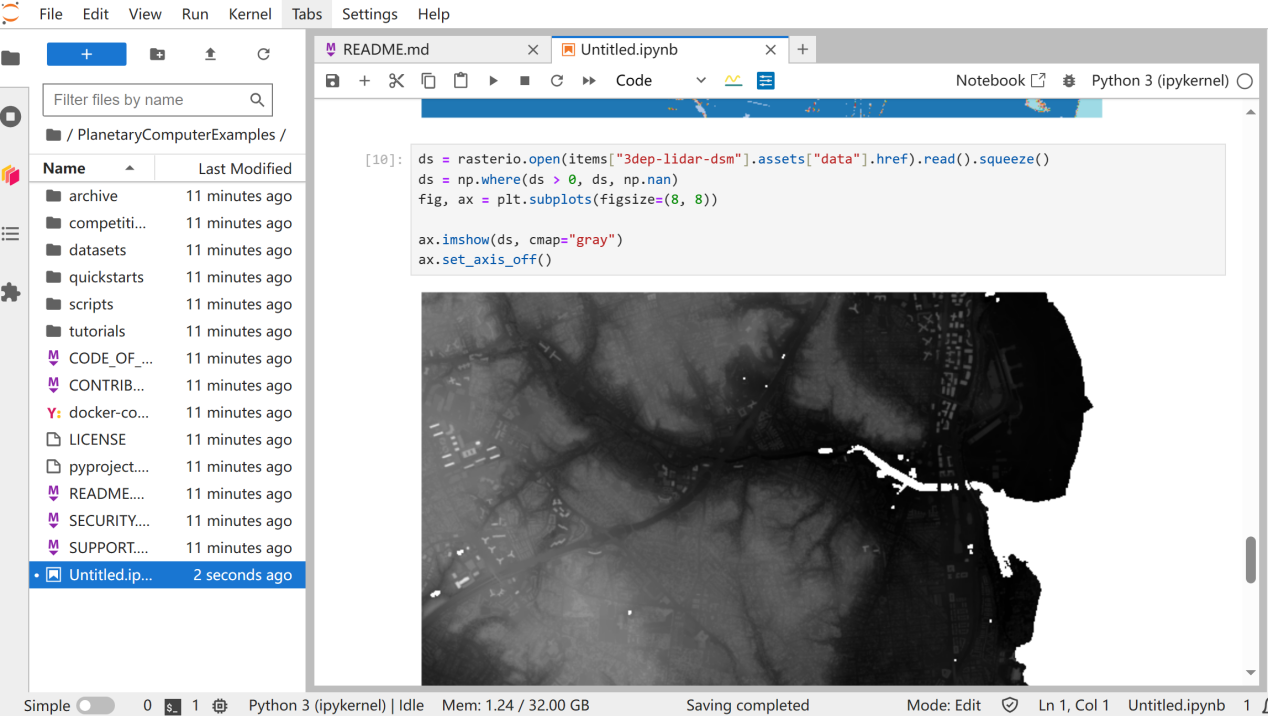
**Classification**

This collection is derived from the USGS 3DEP COPC collection. It uses the ASPRS (American Society for Photogrammetry and Remote Sensing) Lidar point classification. See LAS specification for details.



**DSM**

This collection is derived from the USGS 3DEP COPC collection. It creates a Digital Surface Model (DSM) using pdal.filters.range to output a collection of Cloud Optimized GeoTIFFs, removing all points that have been classified as noise.



**The steps to download the "3dep-lidar-dsm" data set are as follows:**

1. First use the previous search result `items` to obtain the data file link of the "3dep-lidar-dsm" data set:

```python

dsm\_href = items["3dep-lidar-dsm"].assets["data"].href

```

2. Then use the `rasterio` library to open the data file corresponding to this link:

```python

import rasterio

with rasterio.open(dsm\_href) as src:

dsm = src.read().squeeze()

```

Here `src.read().squeeze()` reads the data in the data file and removes any length-1 dimensions.

3. Process the data and replace values less than or equal to 0 with `np.nan` (missing values):

```python

import numpy as np

dsm = np.where(dsm > 0, dsm, np.nan)

```

4. Use the `matplotlib` library to visualize the processed data:

```python

import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(8, 8))

ax.imshow(dsm, cmap="gray")

ax.set\_axis\_off()

plt.show()

```

Here the elevation data is displayed using the grayscale colormap "gray".

So, the complete code is as follows:

```python

import rasterio

import numpy as np

import matplotlib.pyplot as plt

dsm\_href = items["3dep-lidar-dsm"].assets["data"].href

with rasterio.open(dsm\_href) as src:

dsm = src.read().squeeze()

dsm = np.where(dsm > 0, dsm, np.nan)

fig, ax = plt.subplots(figsize=(8, 8))

ax.imshow(dsm, cmap="gray")

ax.set\_axis\_off()

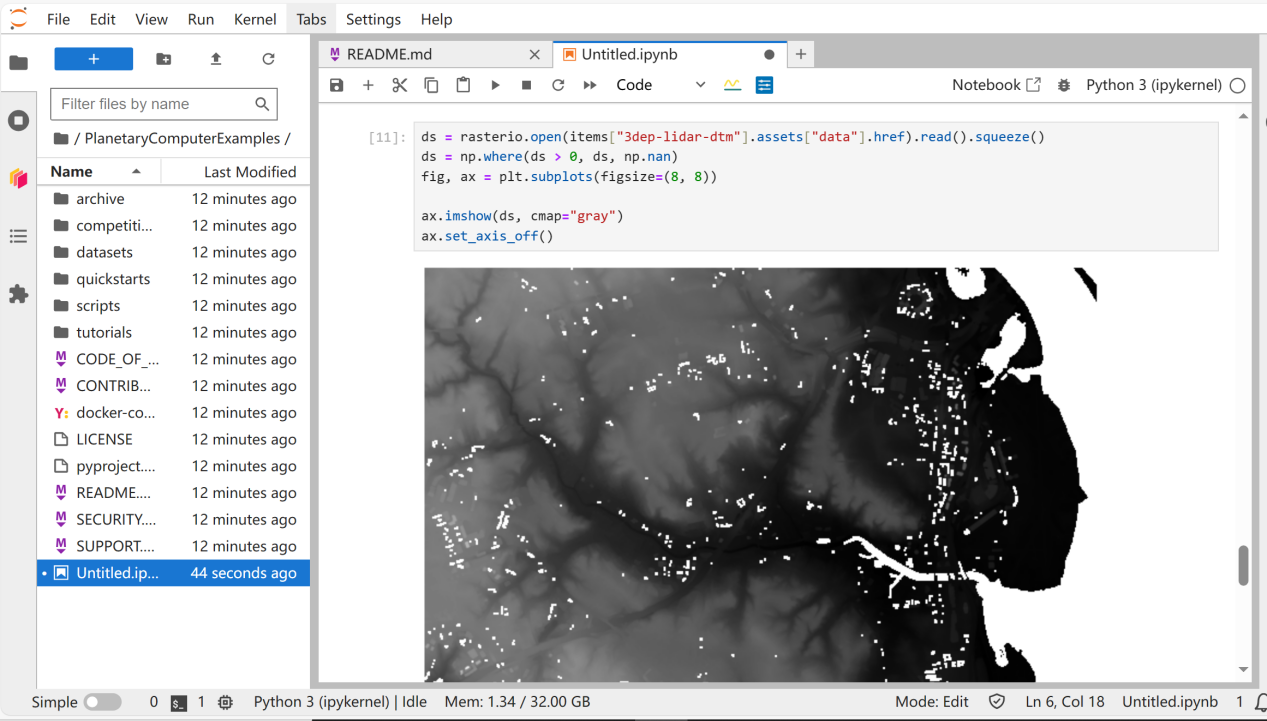
plt.show()

```

This code first obtains the link to the "3dep-lidar-dsm" data set from the search results, then uses the rasterio library to read and process the data, and finally uses matplotlib to visualize the processed data.

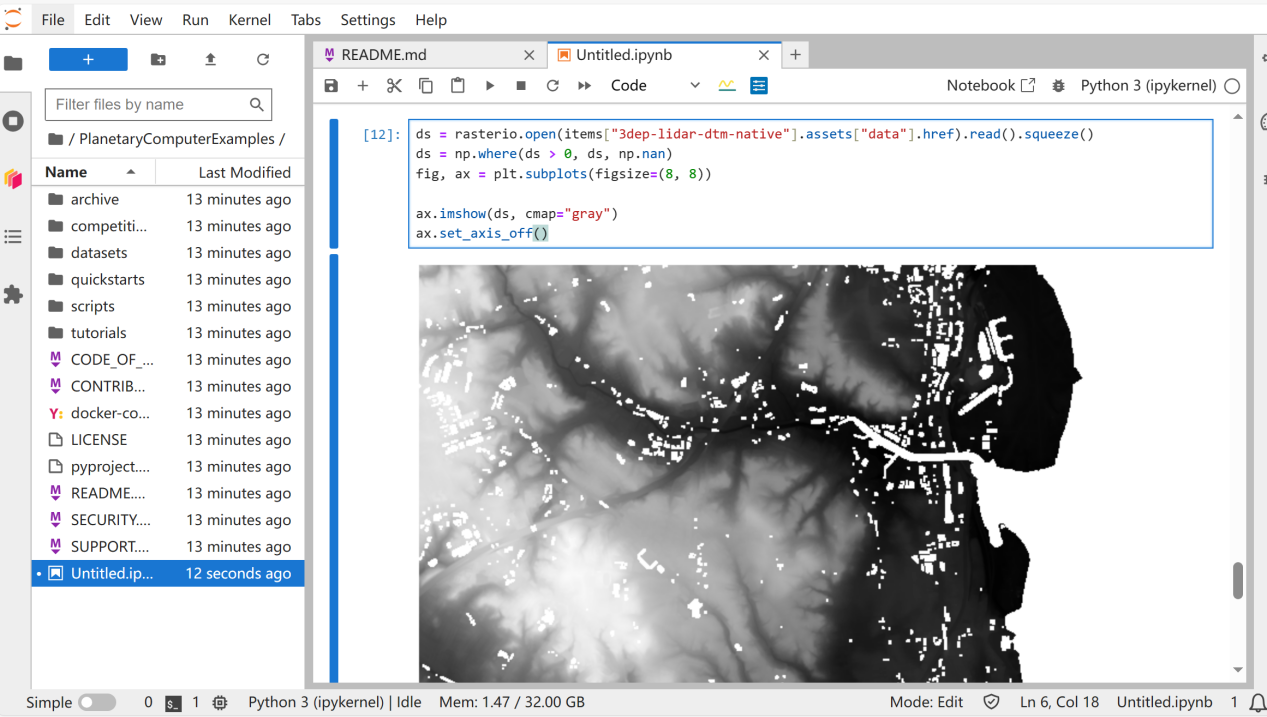
**DTM**

This collection is derived from the USGS 3DEP COPC collection. It creates a Digital Terrain Model (DTM) using pdal.filters.smrf to output a collection of Cloud Optimized GeoTIFFs.



**DTM Native**

This collection is derived from the USGS 3DEP COPC collection. It creates a Digital Terrain Model (DTM) using the vendor provided (native) ground classification and pdal.filters.range to output a collection of Cloud Optimized GeoTIFFs, removing all points that have been classified as noise.



**Point Source ID**

This collection is derived from the USGS 3DEP COPC collection. It is a collection of Cloud Optimized GeoTIFFs representing the file source ID from which the point originated. Zero indicates that the point originated in the current file.

