Methane Emission Sensing Tools - How To Guide

Nicholas Kinsella

Ulster University

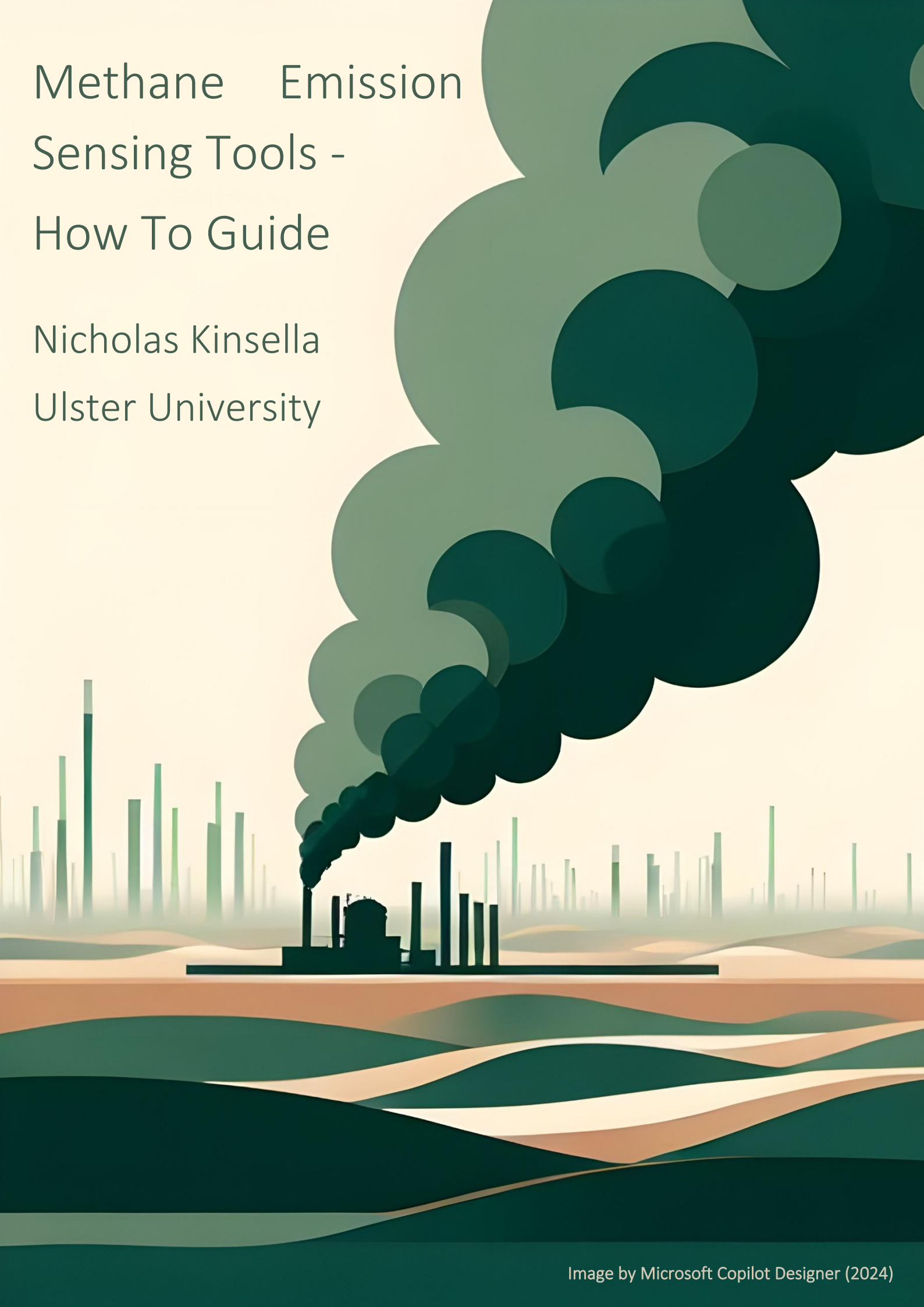

Image by Microsoft Copilot Designer (2024)

Table of Contents

1. Introduction

Methane (CH_4), a greenhouse gas with a warming potential 28 times greater than carbon dioxide over a 100-year period, has seen its atmospheric concentration increase over 250% since the industrial revolution. Despite CH_4's shorter atmospheric lifespan, it still has contributed to at least a quarter of anthropogenic warming since the Industrial Revolution (Pandey et al., 2023; Vigano et al., 2008).

Mismanaged oil and gas fields can produce significant CH_4 emissions (Jackson et al., 2020; Pandey et al., 2023) the 2021 Global Methane Pledge (GMP), was created to reduce anthropogenic emission levels by 30% of 2020 levels by 2030, with 157 countries participating (GMP, 2024; International Energy Agency, 2022; Malley et al., 2023).

In 2022 Algeria made a pledge to reduce its GHG emissions by between 7% and 22% (United Nations Development Programme, 2022) and the European Union has plans to launch a pilot scheme to encourage the Algerian state petrochemical company Sonatrach, to capture CH_4 under the 'You collect, we buy' scheme by the end of 2024 (European Commission,2023). Despite this, as of 2024, Algeria has yet to join the GMP and currently does not have government led commitments to reduce CH_4 emissions specifically (International Energy Agency, 2024).

Managed by state-owned company, Sonatrach, there are well over a hundred identified oil and gas fields in Algeria, two of which are considered giant (Abada et al., 2018). Situated in northern Algeria, 550 km south of Algers (fig.1), the Hassi R'Mel gas field covers an area of around 2,900km^2 and is a key resource for Europe, producing around 45% of Algeria's national gas production (Abada and Bouharkat, 2018; Naus et al., 2023; Rosenthal., 2023; Talamali., 2016). A further 350 km further to the southeast lies the Hassi Messaoud oil field, which covers 2,250 km^2 and produces around 36% of Algeria's national oil production (fig.1) (Kamr Eddine Aissou., 2024; Naus et al., 2023). Despite being an oil field, Hassi Messaoud is by far the worse emitter of CH_4 emissions, with Sentinel 5P detecting 154 super emissions since 2019, compared to just 8 in Hassi R'Mel over the same period, (Kayrros, 2024; Naus et al., 2023). This is backed up by the study of Naus et al. (2023) which detected 1 super-emitter event in Hassi R-Mel and 9 in Hassi Messaoud for the year 2020. Another study by Varon et al. (2021) detected 101 plumes between the 9th of October 2019 and 9th of August 2020 at Hassi Messaoud. This implies that current management practices of CH_4 emissions at Hassi Messaoud could be improved.

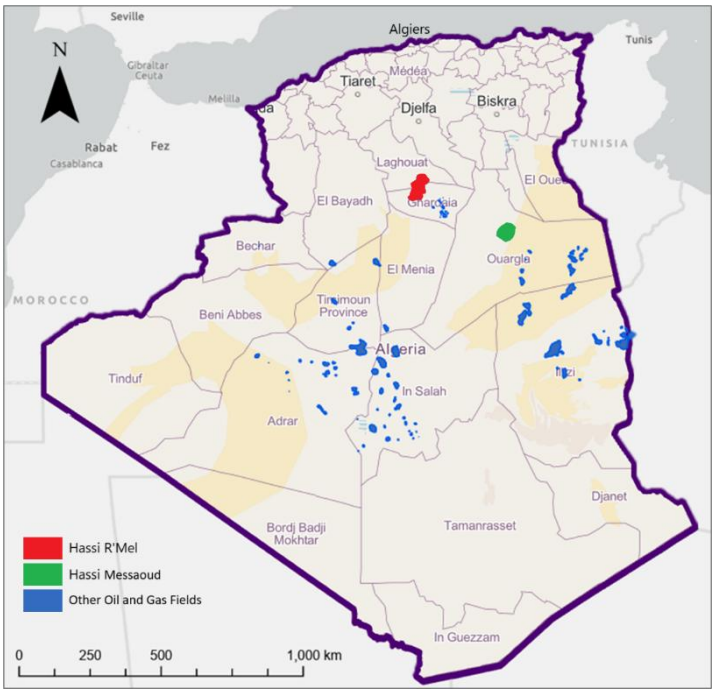

Figure 1: Hassi Messaoud oil field in Green (Lat 31.69, Long 6.03) and Hassi R'Mel gas field in Algeria in red (Lat 32.94, Long 3.27). (Field locations produced by manual survey of satellite images by author.)

Satellite missions have in recent years improved their CH_4 measurement capabilities, offering advantages over ground-based detectors (Parker et al., 2011). This guide outlines the use of two tools that can be used to monitor methane emissions.

- **The Integrated Methane Inversion (IMI):** An Amazon Web Services based tool for measuring methane emissions using the Sentinel-5P methane product and the GEOS-Chem model of atmospheric chemistry (Varon et al., 2022). Full details of the tool can be found here: https://imi.readthedocs.io/
- **Sentinel-2 Plume Finder:** This creates a high-resolution map that can show super emitting CH_4 point sources for selected Algerian petrochemical fields for a specific date. It will also provide an estimation of the emission rate in kg/h. This tool can be downloaded or cloned from the following Github repository: https://github.com/zelcon01/Sentinel-2_Algeria_Methane/

This guide outlines their installation and use.

2. Integrated Methane Inversion (IMI)

2.1 Methodology

The Sentinel-5P Tropospheric Monitoring Instrument (TROPOMI) Methane product provides atmospheric CH4 concentrations in parts per billion (ppb). Its data has been shown to be in close agreement with ground-based measurements from the Total Carbon Column Observing Network (Liu et al., 2021; Lorente et al., 2021). Additional steps are required to estimate ground level CH4 emissions from these observed whole atmosphere concentrations, because CH4 can travel significant distances from its source and may be present in high altitudes over otherwise low-emission areas, and vice versa (Varon et al., 2022), resolving this requires an atmospheric transport model to be applied the data

The IMI (Integrated Methane Inversion) is a Python-based tool developed by Varon et al. (2022) on Amazon Web Services to estimate emissions in a selected area (fig. 2). Official bottom-up emission inventories of CH_4 are known to often be inaccurate (Dowd et al., 2023; Elkind et al., 2020; Karimi et al., 2021; Sherwin et al., 2024; Vaughn et al., 2018; Wang et al., 2024). The IMI seeks to improve on the bottom-up inventories by comparing them to top-down measurements by Sentinel -5P. Thereby providing a more accurate figure of emissions than official figures.

The inversion process starts with a "prior estimate of CH_4 emissions" based on official records such as the Global Fuel Exploitation Inventory (Scarpelli et al., 2022). Like all gasses CH_4 is affected by weather and other atmospheric influences, so next an atmospheric transport model is applied to the prior estimate data to create a model of predicted atmospheric concentrations, that should be seen by Sentenel-5P. The prediction is then compared to Sentenel-5P's real observations. The "inversion" involves adjusting the predicted concentrations so that they better match the observed concentrations. This results in a "posterior estimate" which should be a better reflection of reality.

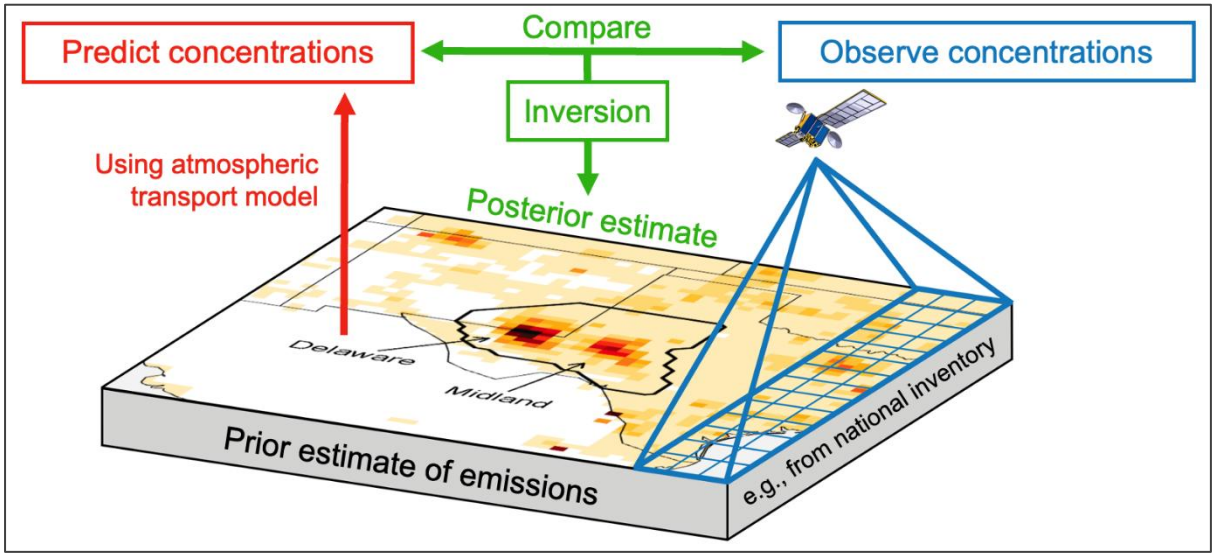

Figure 2: IMI processing step (Integrated Methane Inversion, 2024).

2.2 Setup

The following steps will show you how to setup a AWS account and run the IMI tool

2.2.1 Creating an AWS account

Creating an AWS account requires providing some personal and bank card information, as it is not a free service. The cost of running the IMI will be provided as an estimate prior to you running the full analysis. Open the following link in a browser http://aws.amazon.com/ and click on create an AWS account (fig. 3). Thereafter, follow the prompts.

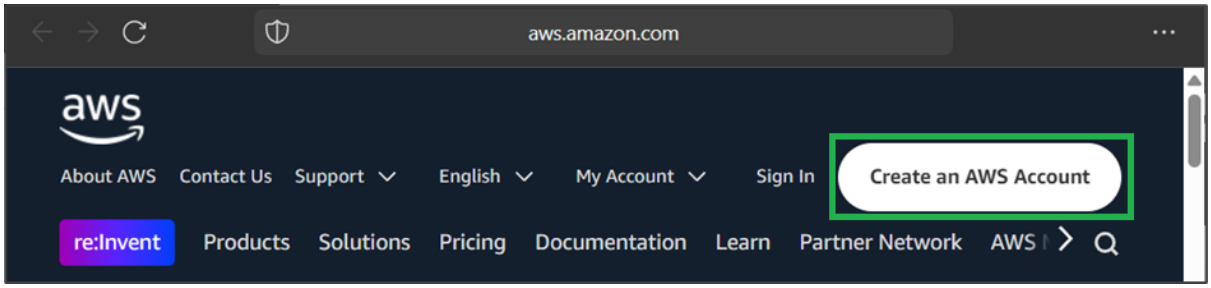

Figure 3: AWS sign in and create account landing page.

2.2.2 Add Amazon Simple Storage Service (S3) user permissions

This is an optional step if you wish to store your results long term. Firstly navigate to the AWS management console by clicking on the "Sign In to the Console" button or by using the "My Account" dropdown (fig. 4).

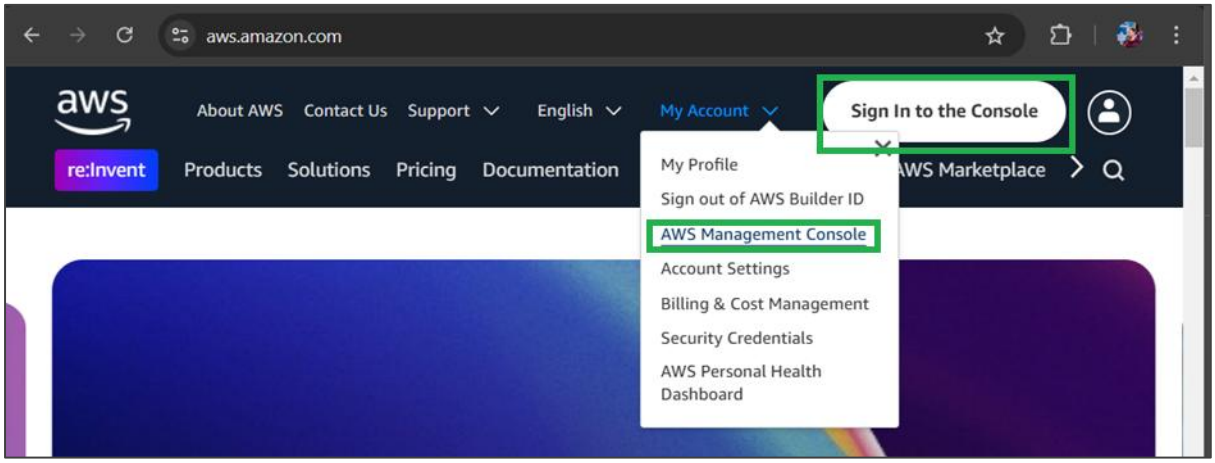

Figure 4: Sign in to my Console access on AWS landing page.

Once on the management console page, click the magnifying glass (fig. 5)

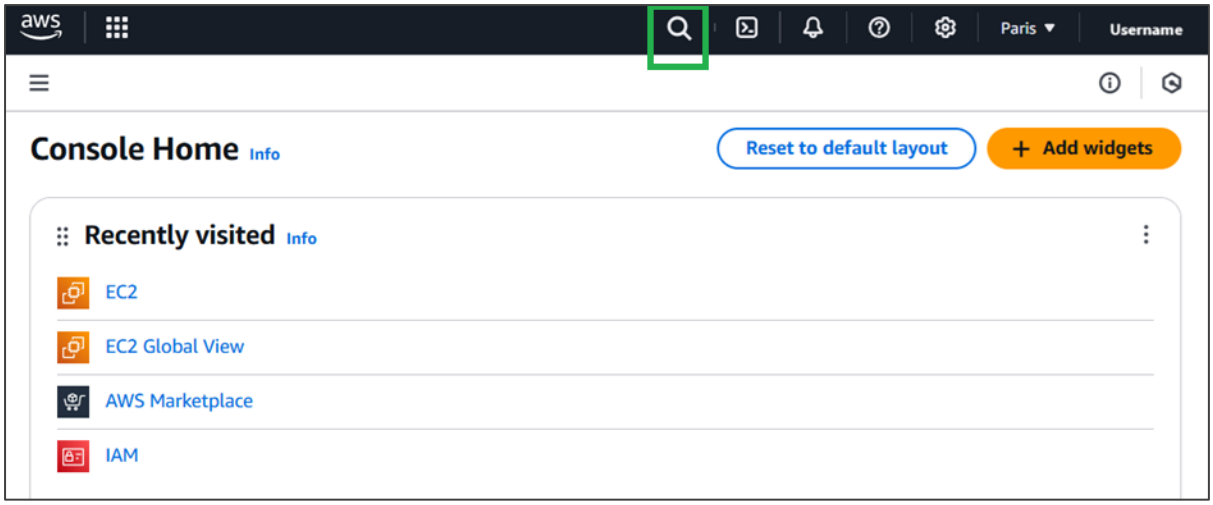

Figure 5: Location of search function in AWS console.

Then search for "IAM" (fig. 6).

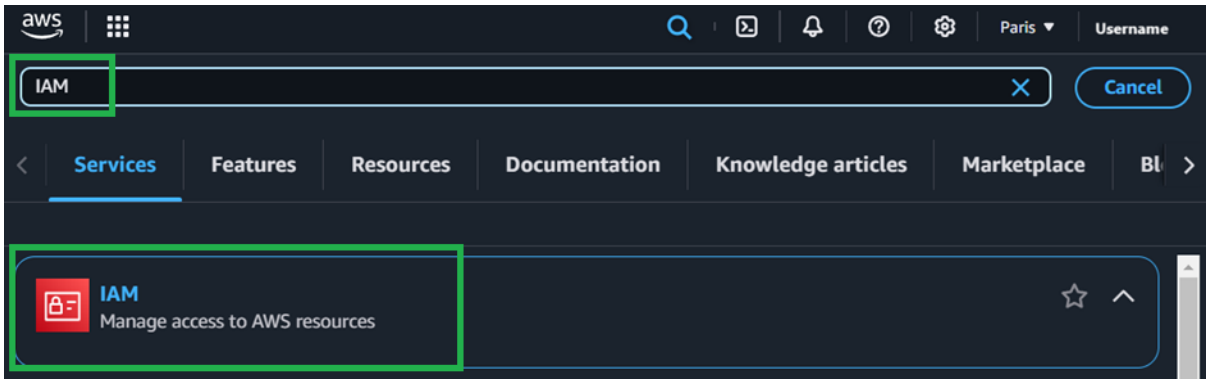

Figure 6: search function finding the AWS resource management access.

Under the "Access management" sidebar, click on Roles and then "Create role" (fig. 7)

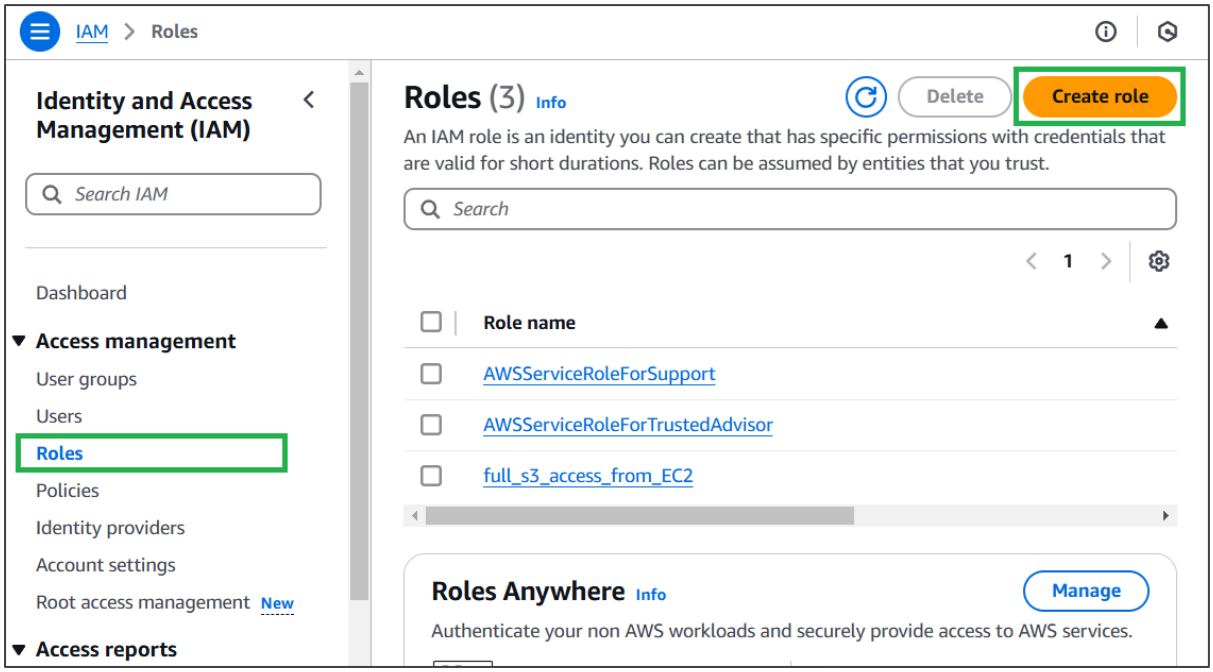

Figure 7: create role button location

In the “Use Case”, select EC2 and then click “next” (fig. 8).

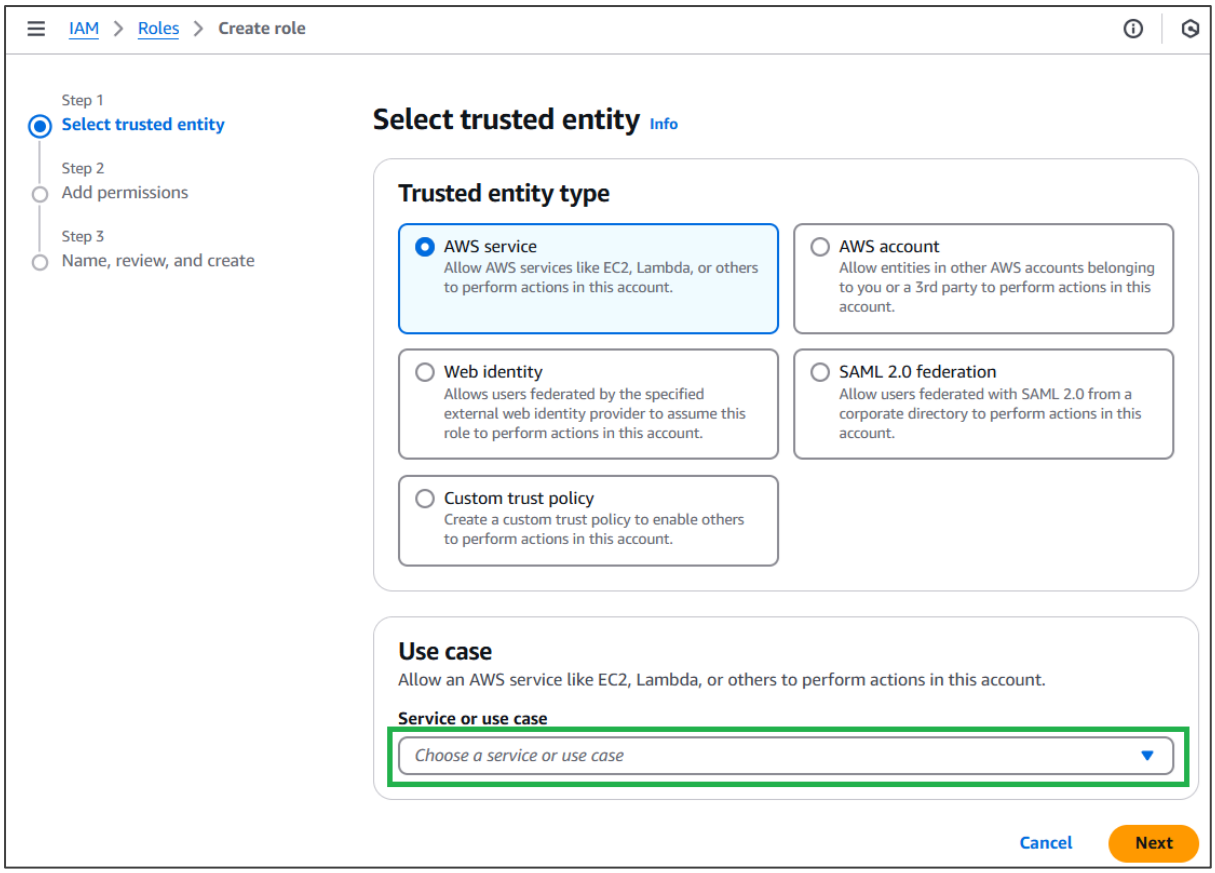

Figure 8: Use case selection location.

In the “Add permissions” screen type S3 in the search box and select AmazonS3FullAccess (fig. 9). Click next.

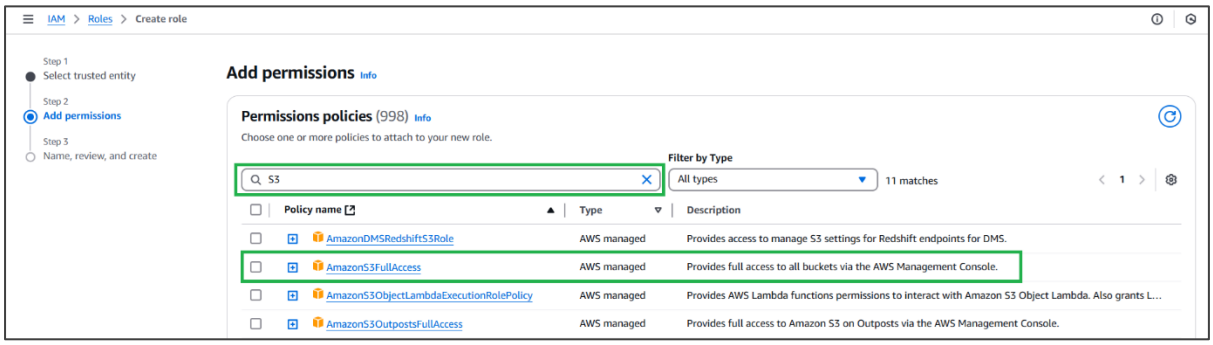

Figure 9: Use case selection location.

Lastly, choose a descriptive name for the role like “full_S3_access_from_EC2”. Click “create role” at the bottom of the page (fig. 10).

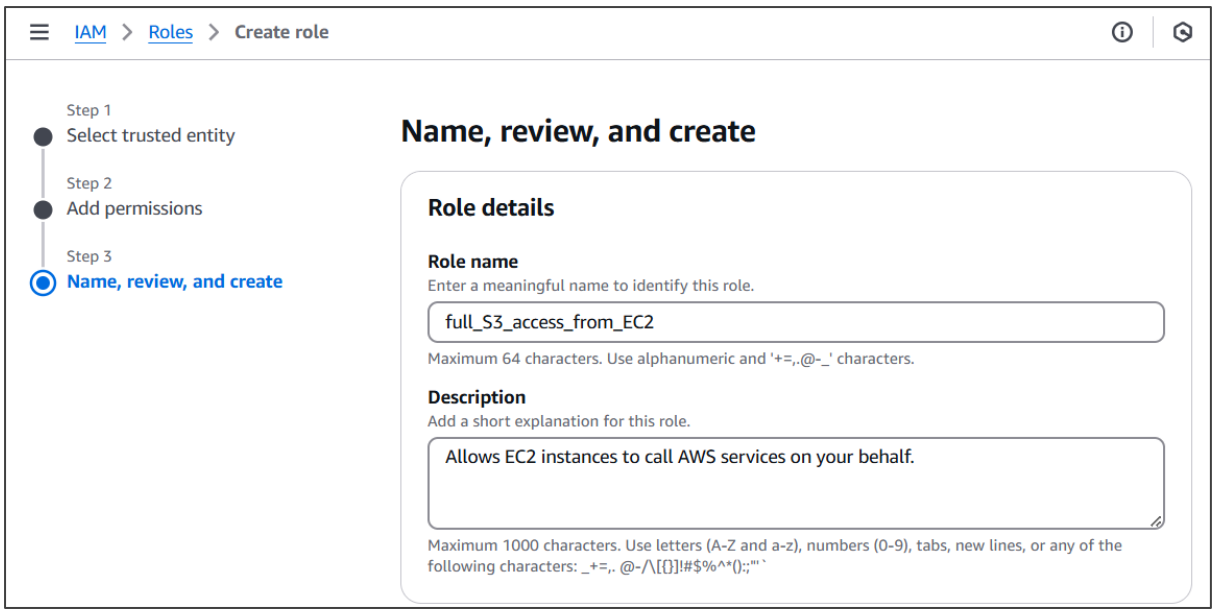

Figure 10: Create roll naming screen.

Now a new IAM role is created.

2.2.3 Launching an IMI instance

Navigate to https://aws.amazon.com/marketplace/pp/prodview-hkuxx4h2vpjba or search for "Integrated Methane Inversion". You should find the page shown in figure 11. Click "view purchase options"

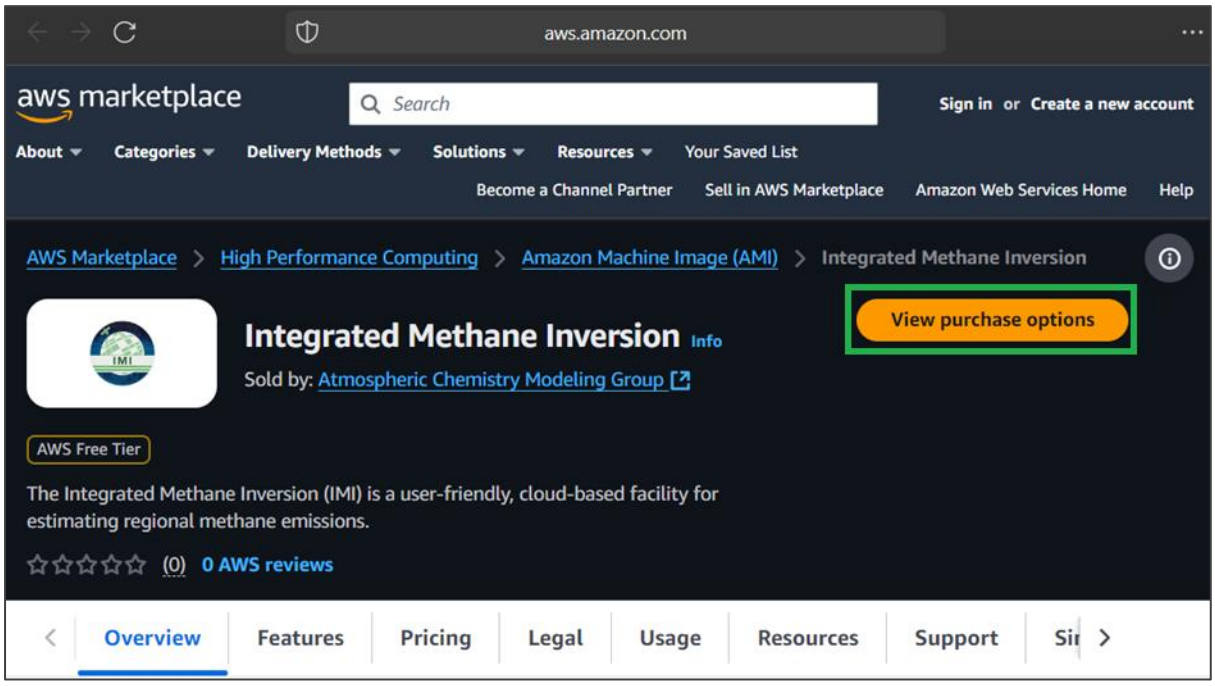

Figure 11: IMI landing page and purchase options button.

Next click "continue to configuration" (fig. 12).

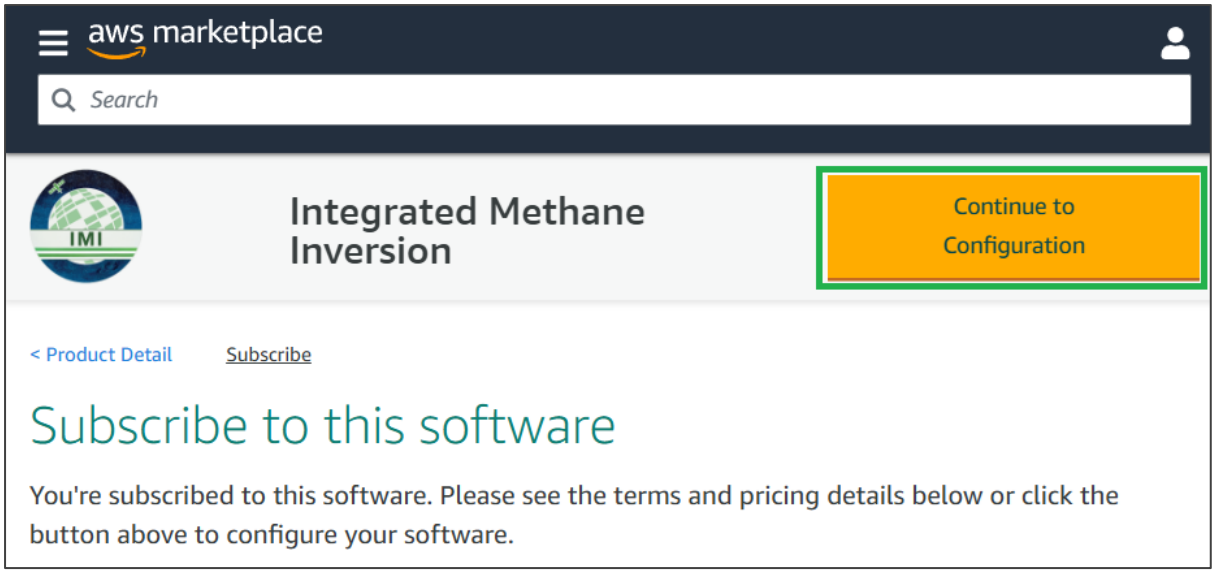

Figure 12: Subscription page with continue to configuration button.

Choose your preferred region and IMI version, then click *"Continue to Launch."* (fig. 13) Opting for a region closer to your physical location can enhance network connectivity but may lead to higher costs compared to selecting the region where GEOS-Chem data is hosted (us-east-1, N. Virginia).

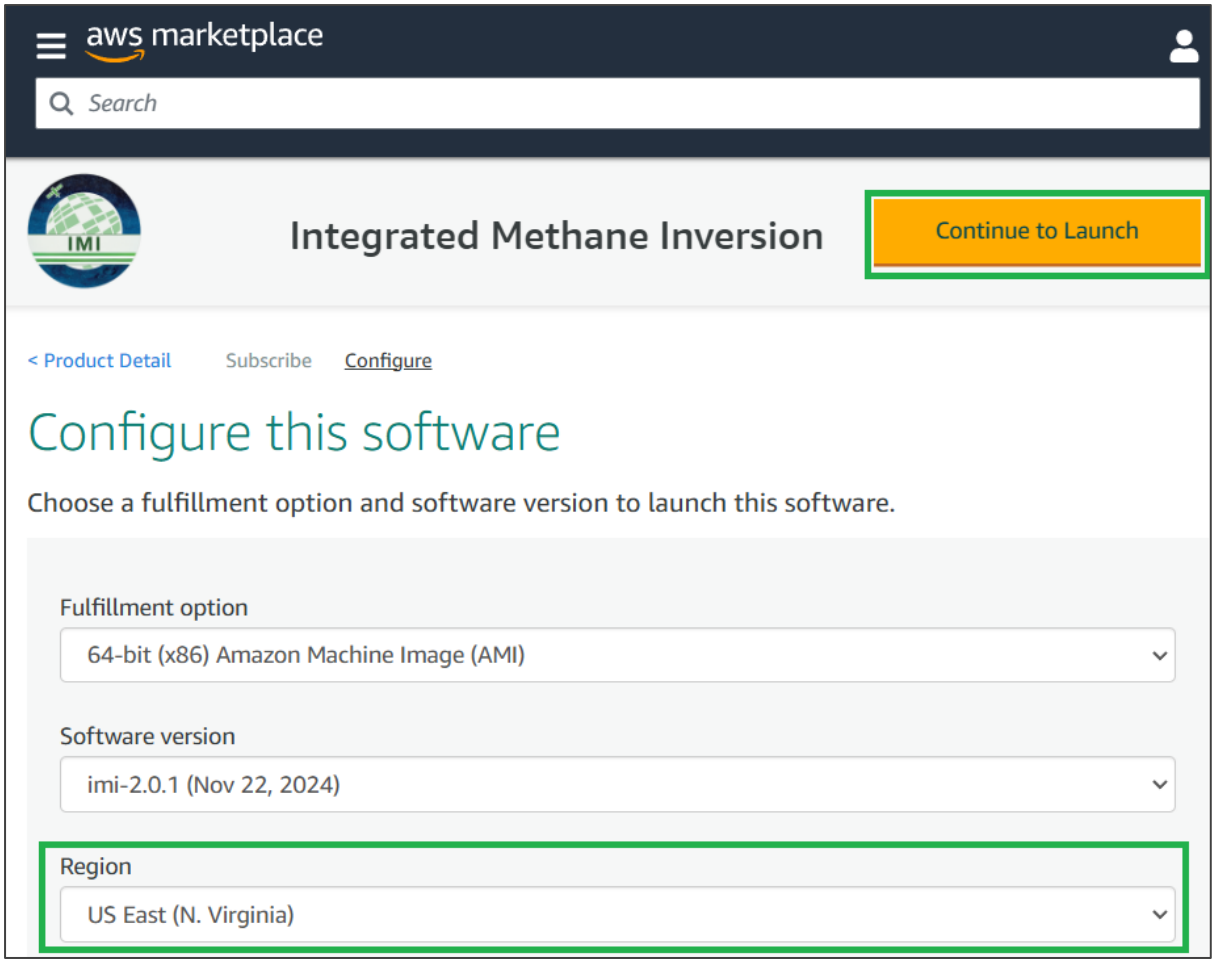

Figure 13: region configuration page and "continue to launch button".

In the "Launch this software" page, in the choose action prompt, select "Launch through EC2" and then click "Launch" (fig. 14).

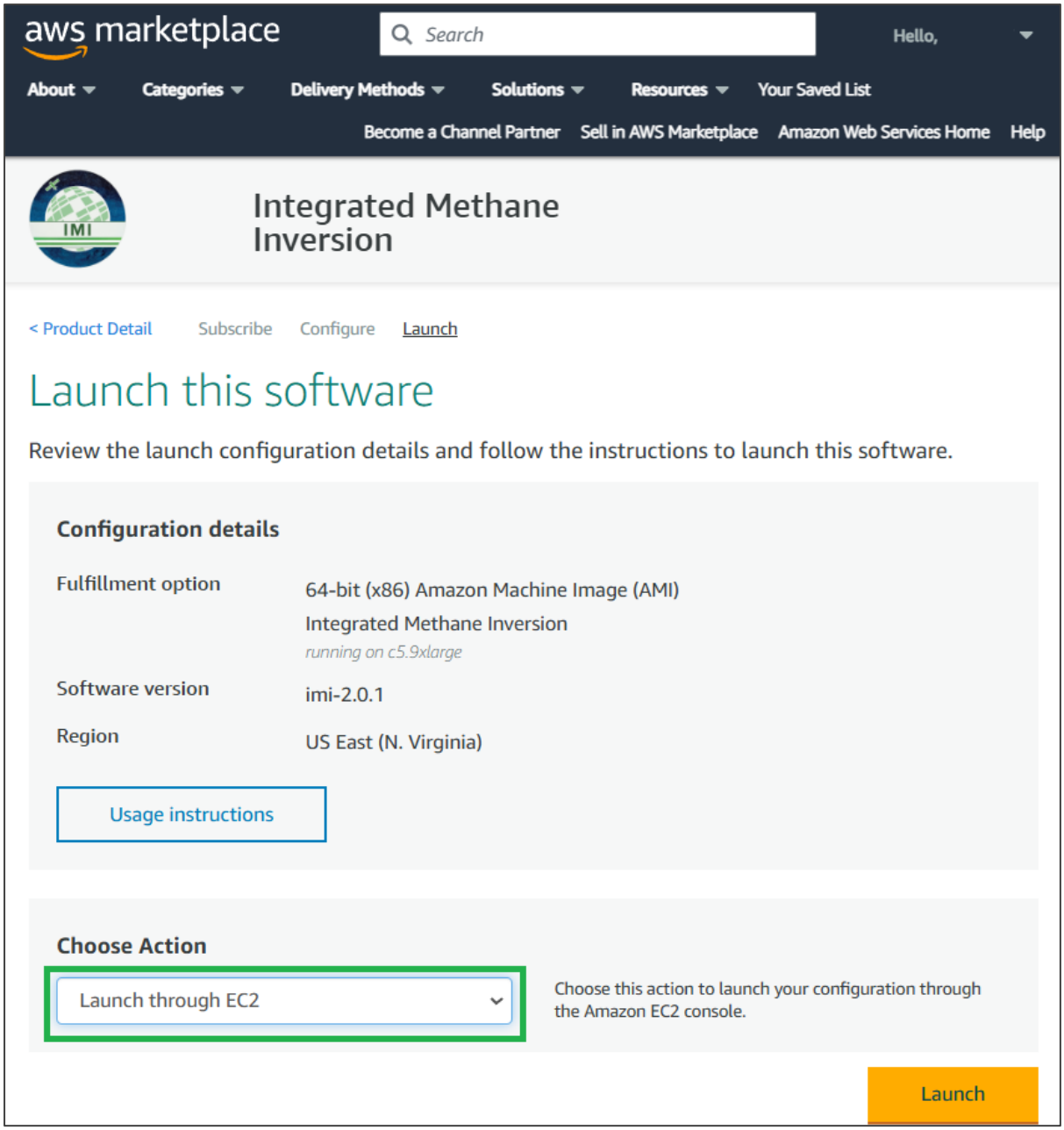

Figure 14: Launch software page.

Now it's time to determine the name of your instance and the hardware to run the IMI. The primary considerations are the number of CPUs and the amount of RAM. In the "Instance Type" section, you can choose from a wide range of options. The IMI will perform faster with a higher number of CPUs. The c5.9xlarge instance type is the recommended setting. It offers 36 CPU cores and 72GB of RAM. However, you can choose a different instance type with more or fewer cores and memory based on your specific needs (fig. 15).

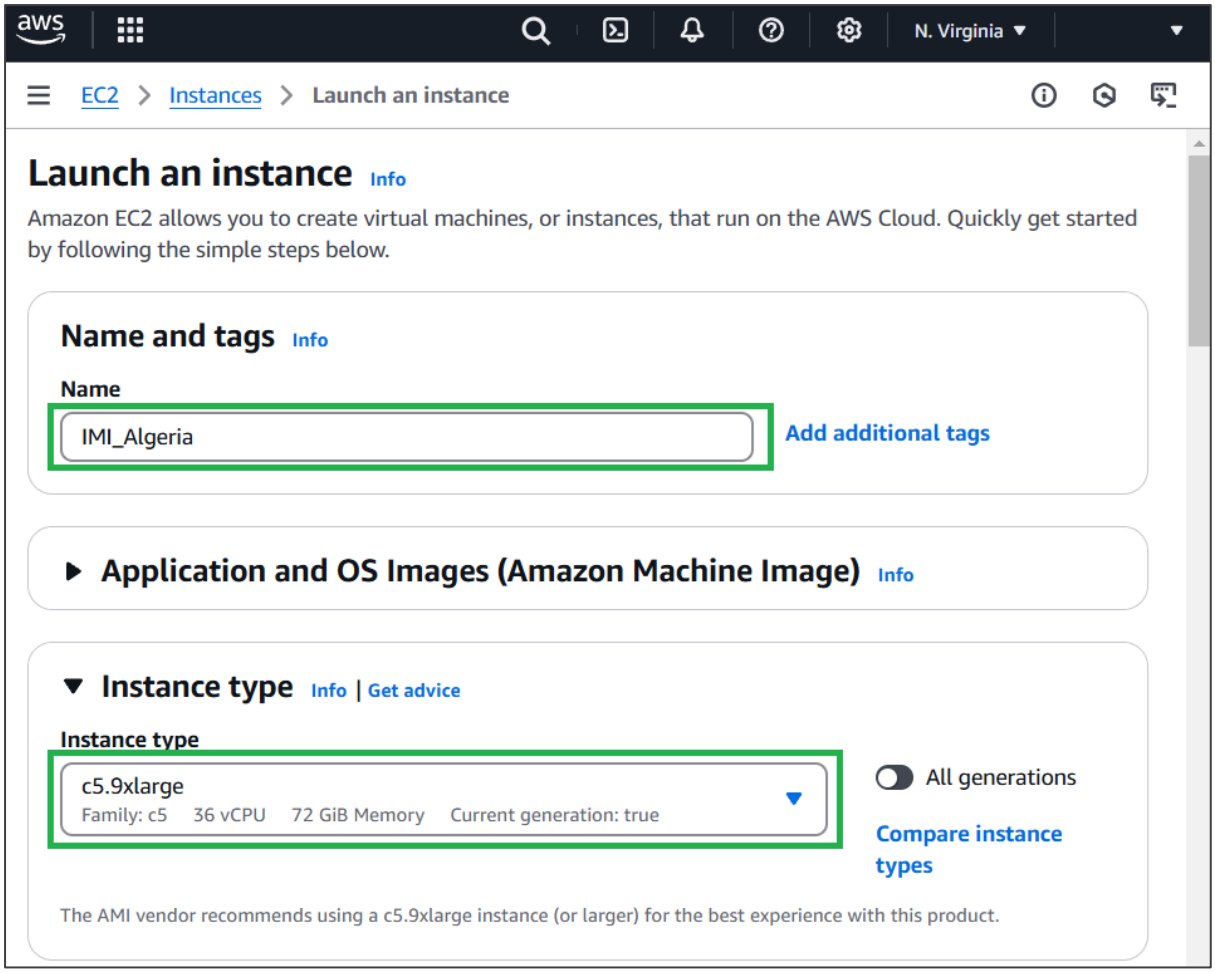

Figure 15: Hardware selection page.

In the next step, you'll create a new SSH key pair or select an existing one. This key pair acts as the password for accessing your server via SSH. To create a new key pair, click “Create new key pair”. In the dialog box, assign a name to your key pair (e.g., imi_testing) and click “Create key pair”. Once created, the key pair will be available for future use, and you can simply select it from the dropdown menu (fig. 16).

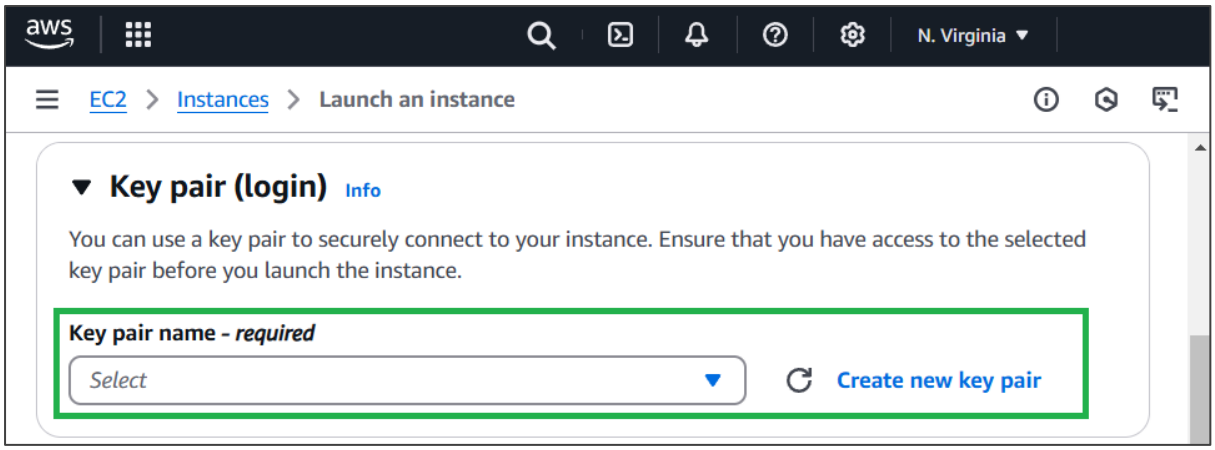

Figure 16: keypair selection.

If you are creating a key pair, use the settings shown in figure 17, choosing an appropriate name. Click “create key pair” once you have finished, saving it in a memorable directory that will be used later.

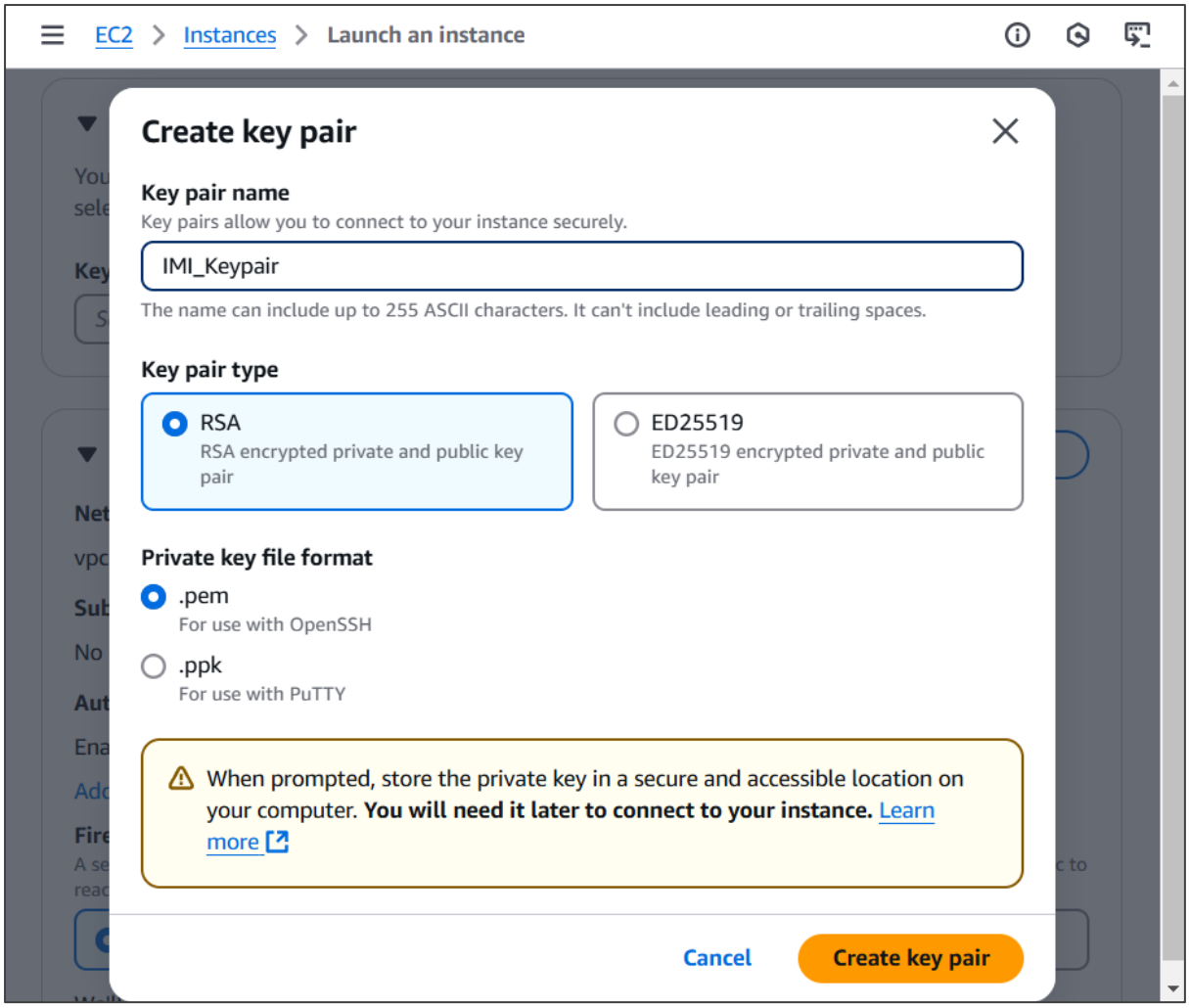

Figure 17: keypair setup.

Proceed to the "Configure Storage" section (fig.18), where you will select the size of the storage volume. Storage costs are around USD $100 per month per TB of space. The amount of storage you choose will depend on how long you intend to analyse an area. According to an example provided by the IMI team, 100 Gibibytes is sufficient for a 1-week inversion of an area of around 10,000km^2, whereas 5 terabytes is enough for 1 year. Also, 10,000km^2 is the minimum recommended area of analysis.

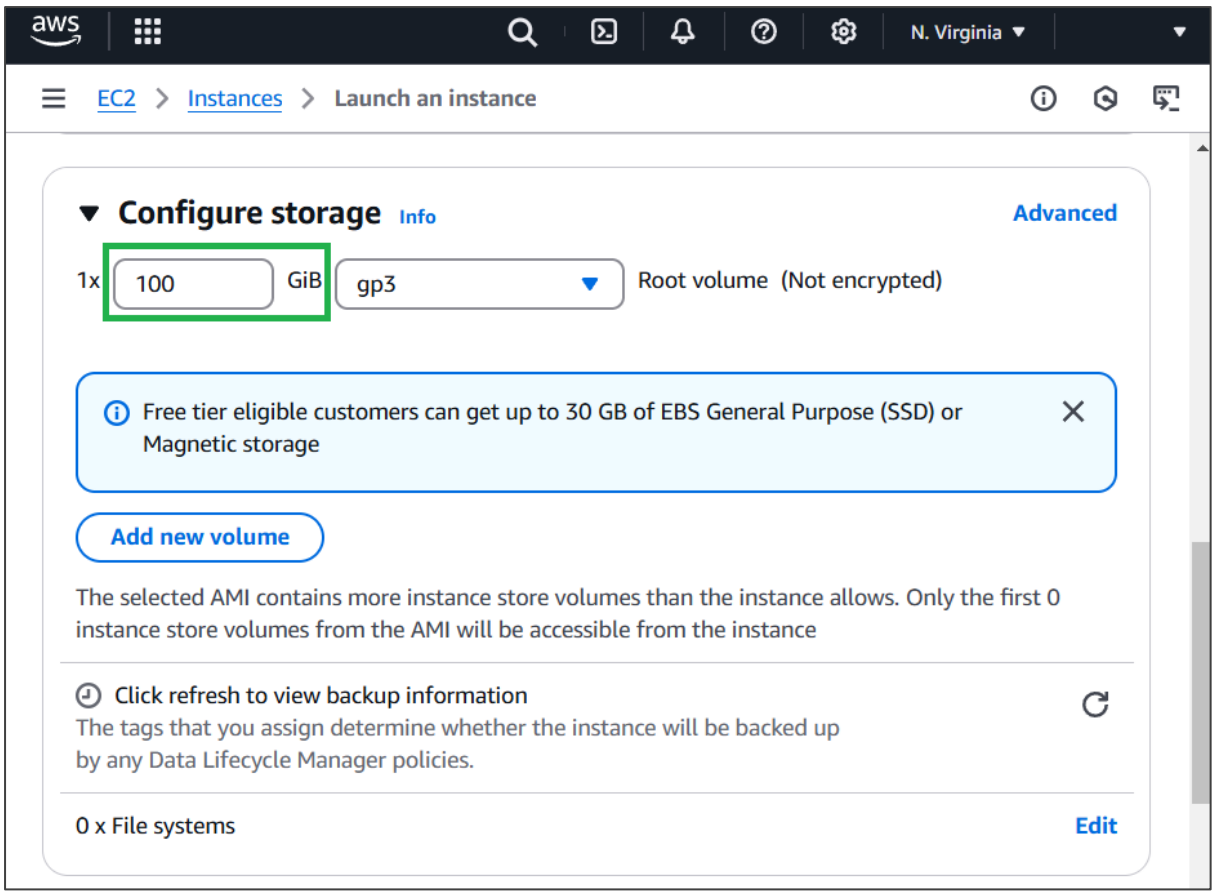

Figure 18: storage configuration section.

Next, open the "Advanced Details" section (fig. 19) and choose the IAM role you created in step 3 from the "IAM Instance Profile" dropdown. This will grant your EC2 instance access to S3 for uploading output data. You can leave all other configuration settings in the "Advanced Details" section as the default values.

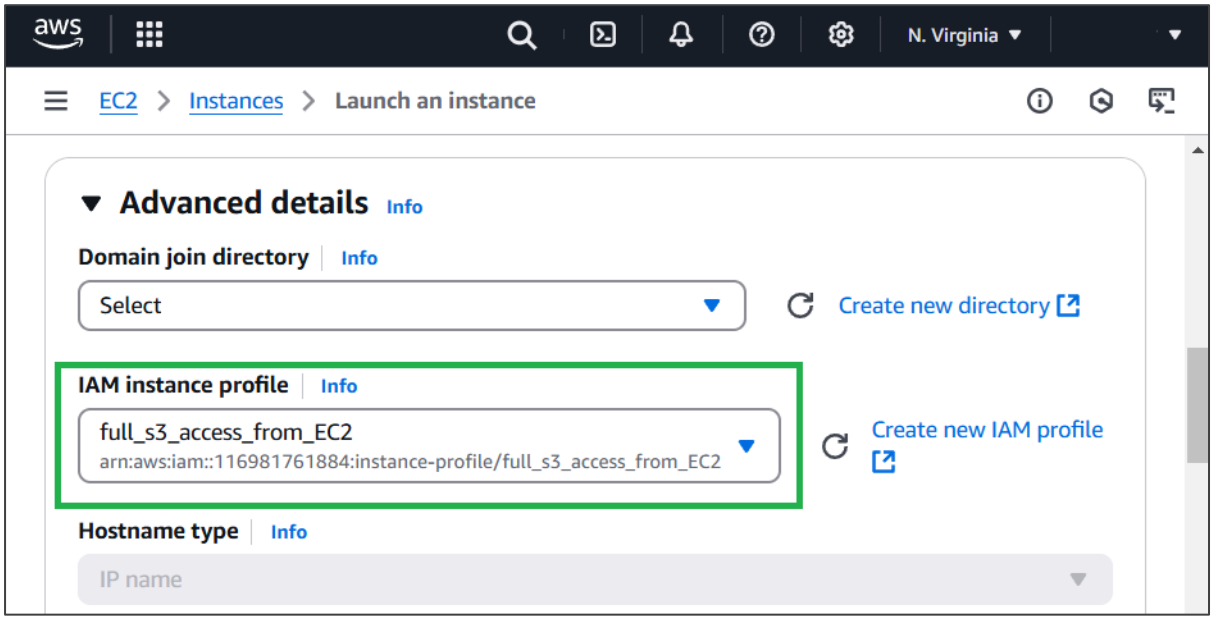

Figure 19: advanced details section.

Finally click the launch instance button in the section showed in figure 20.

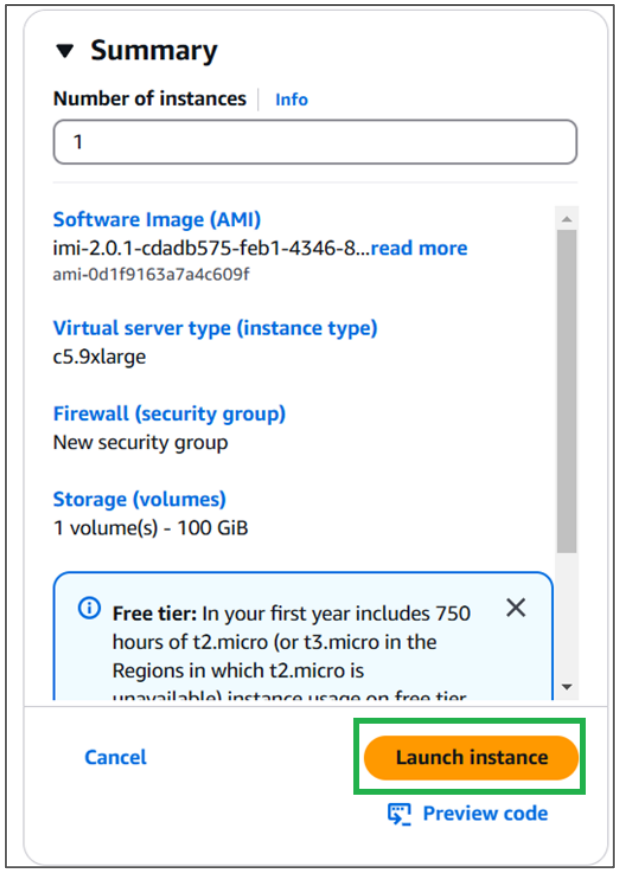

Figure 20: Summary section with "launch instance" button.

2.2.4 Connecting to instance

To connect to the instance and run the IMI, a program called Git will need to be downloaded and used. You can download Git from the following URL: https://git-scm.com/downloads/win, choosing the appropriate version of the software for your system (fig. 21).

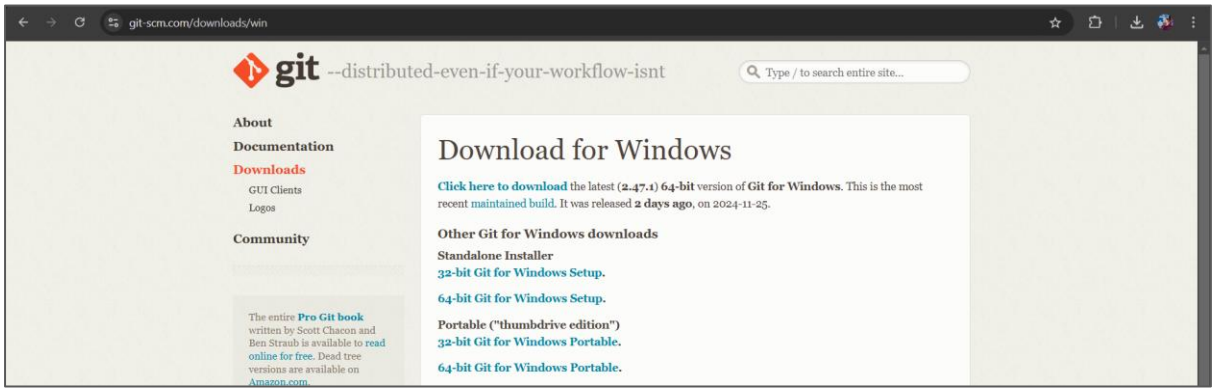

Figure 21: download page for Git.

Once downloaded, run and follow the instructions on the installation wizard (fig. 22).

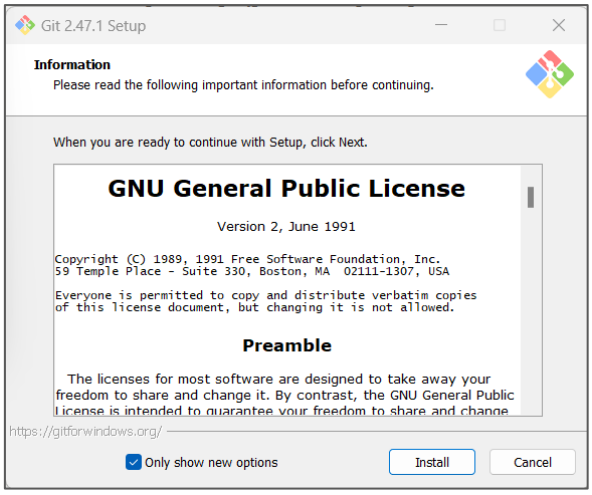

Figure 22: Git installation wizard.

Once completed, tick the box that says launch Git Bash (fig. 23). We will come back to it shortly.

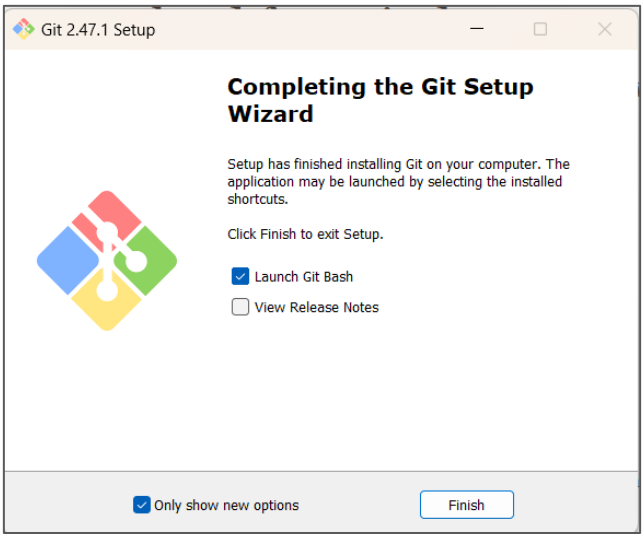

Figure 23: Git installation wizard.

Returning to the AWS management console, the first time you launch an instance it should be running already (fig. 24).

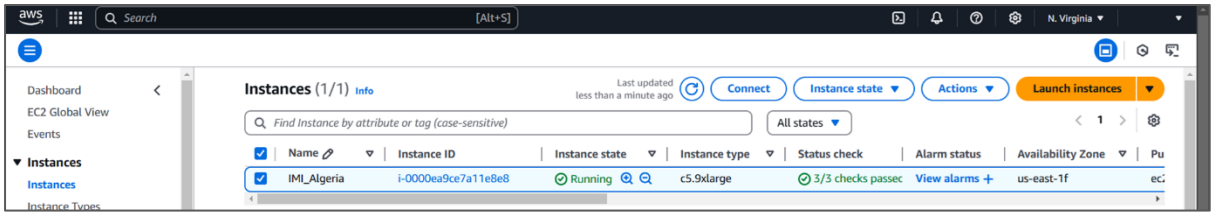

Figure 24: AWS management console Instance dashboard.

If it isn't running then select the instance using the tick box an then under "instance state" select "start instance" (fig. 25).

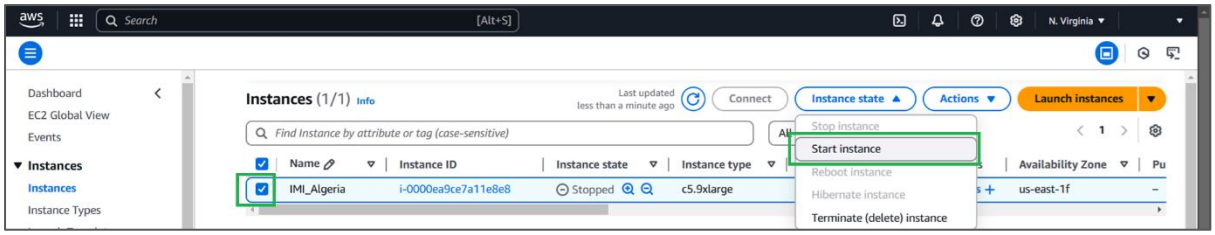

Figure 25: start instance button.

Once running, with the instance selected, select “actions” and then “connect” (fig. 26)

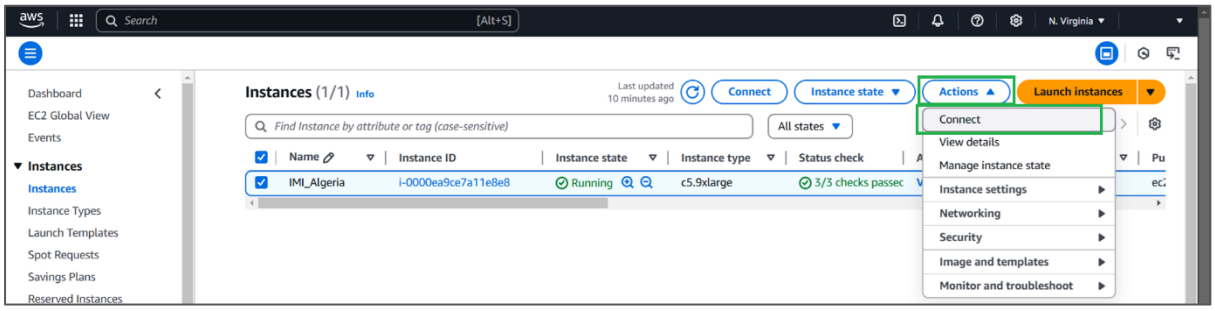

Figure 26: connect button.

In the connect to instance page, you have a list of instructions to follow using the Git Bash terminal you installed (fig. 27).

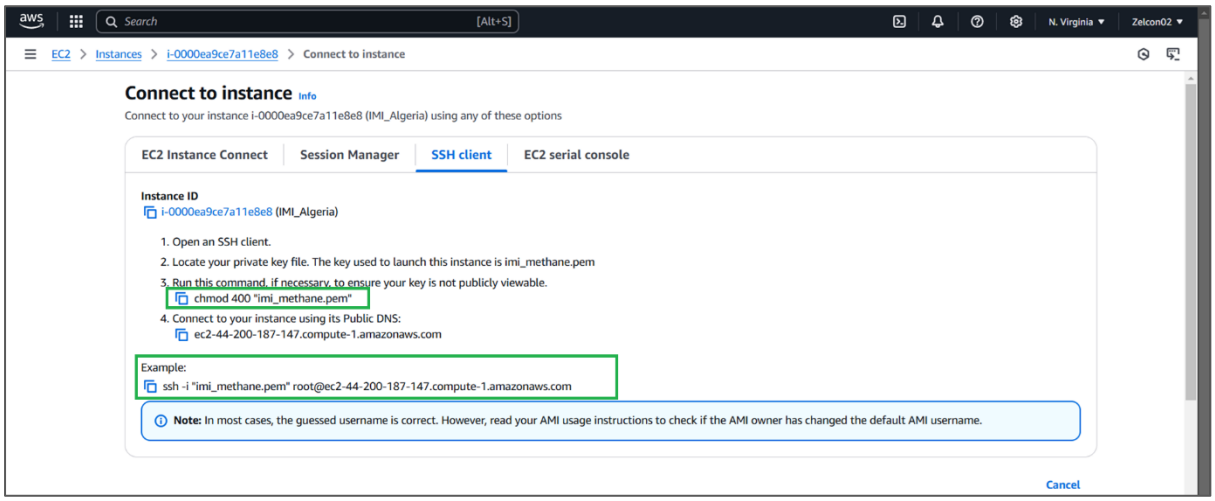

Figure 27: code to prevent keypair from being publicly visible and ssh connection command.

Firstly change directory to where you keep your keypair by typing cd <path to your keypair folder> (fig. 28)

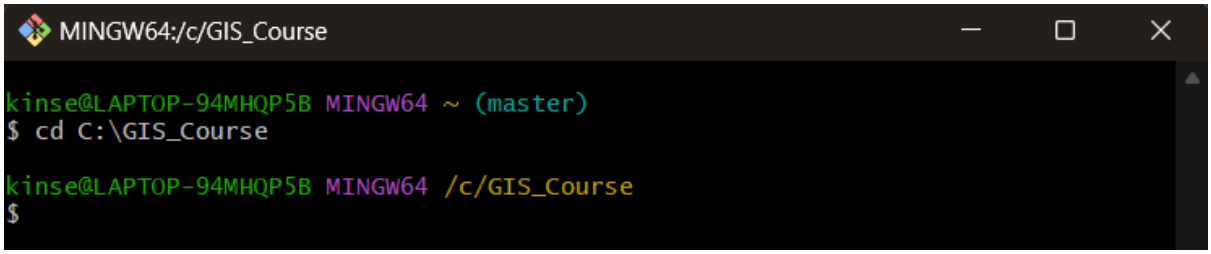

Figure 28: Gitbash console showing path to keypair folder.

Next you are going to enter in the command that ensures your keypair is not publicly viewable (fig. 29).

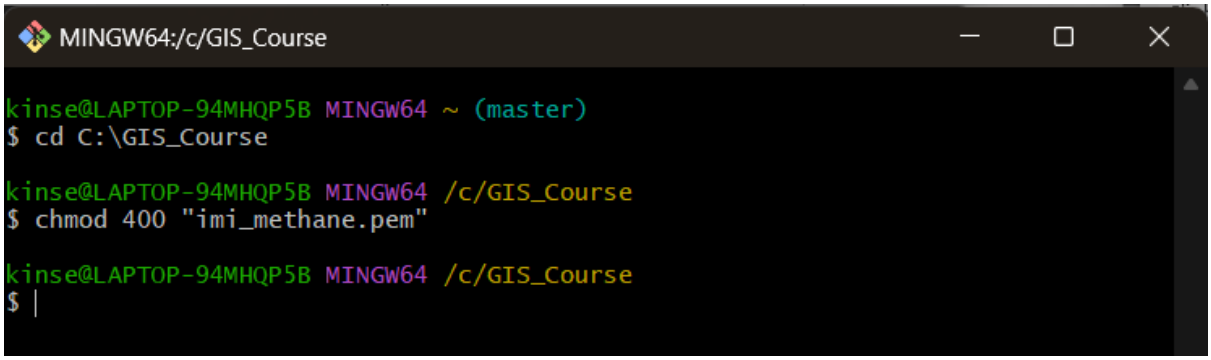

Figure 29: hiding your keypair.

Finally you are going to take the long "example" code (at the bottom of fig. 27) and change the word "root" to "ubuntu".

ssh -i "imi_methane.pem" root@ec2-44-200-187-147.compute-1.amazonaws.com

ssh -i "imi_methane.pem" ubuntu@ec2-44-200-187-147.compute-1.amazonaws.com

Then paste it into the Git Bash terminal using the right click function of the mouse. Run the code and answer "yes" to the prompt, "are you sure you want to continue connecting?" (fig. 30). Once this has run, you should receive a "welcome to ubuntu" message.

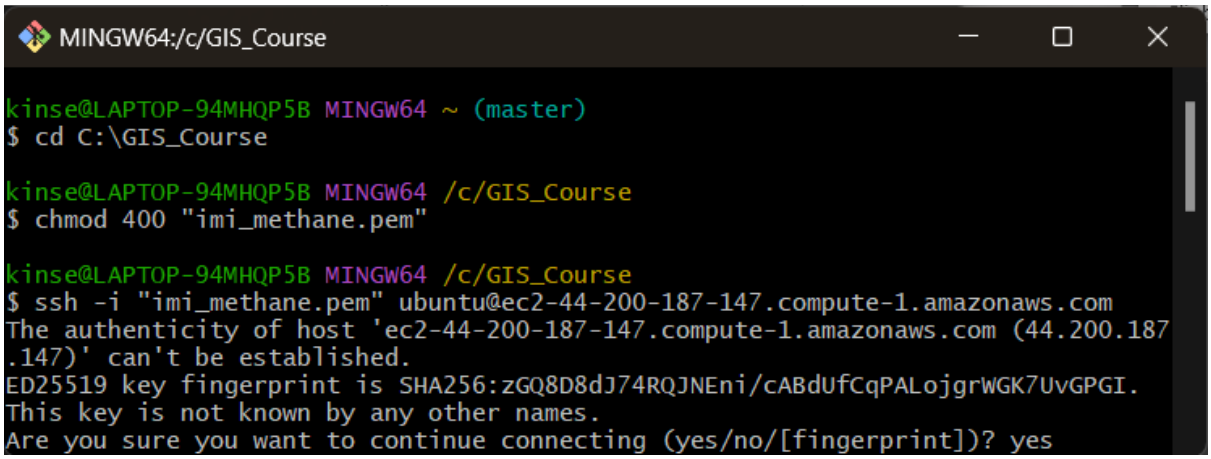

Figure 30: ssh connection to your instance.

2.3 Configuring and running the IMI Preview

In the Git Bash terminal, navigate to the IMI setup directory by typing:

cd ~/integrated_methane_inversion

Next open the inversion configuration file by typing the following:

emacs config.yml

The following screen should appear after a pause of a few moments (fig. 31)

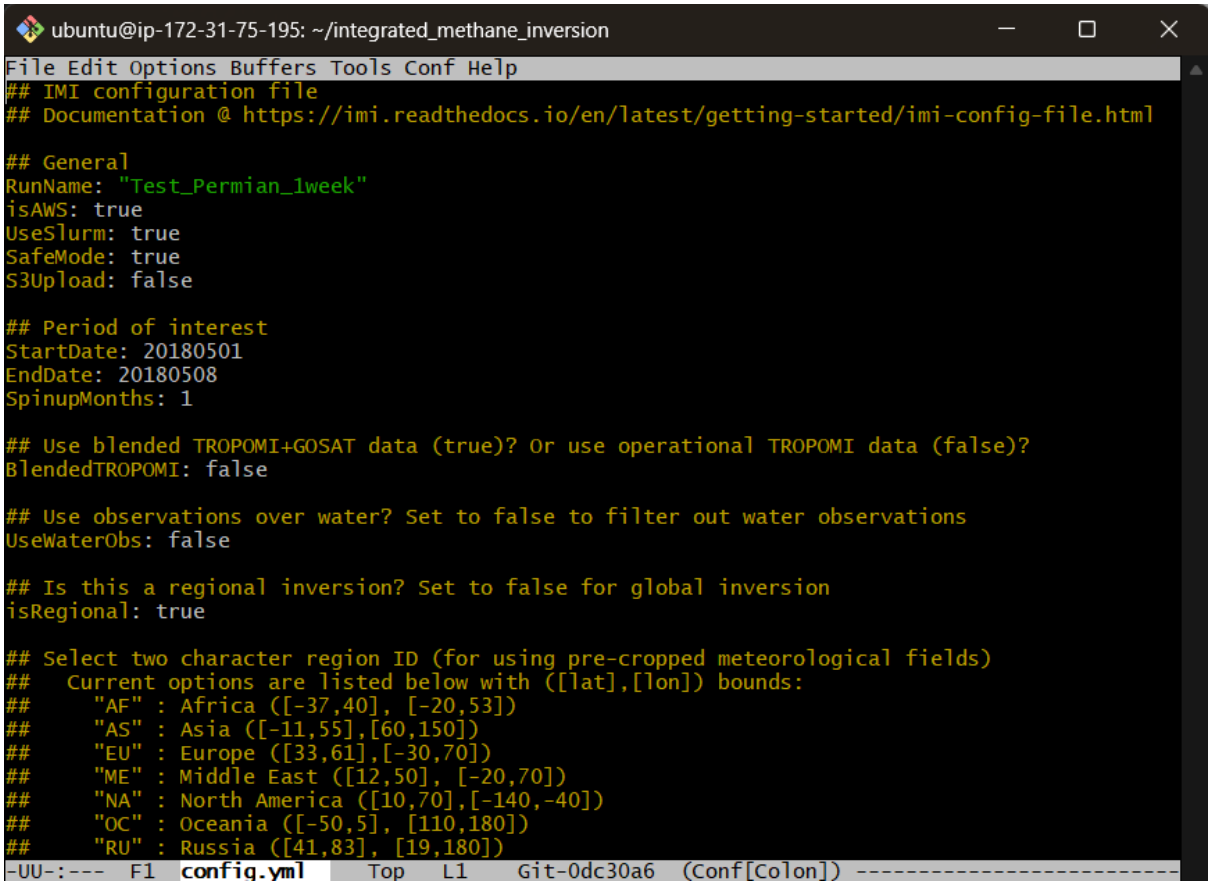

Figure 31: Beginning section of Config.yml

By default, this will run the IMI preview which will provide among other things, the expected cost of running the full inversion.

Choose an appropriate RunName and make a note of it. This must be unique for every inversion you run. Slightly further down, you can modify the date fields (format: YYYYMMDD) to choose the dates for the analysis (fig. 32).

Figure 32: RunName field and start and end dates

Bounding box settings are found further down the page in the "region of interest" section (fig. 33). Specific boundings for each oil and gas field are given in figure 59 and table 1 on page 34 and 35 of this guide. As you are going to be looking at the African continent the RegionID must be set to "AF" (fig.33)

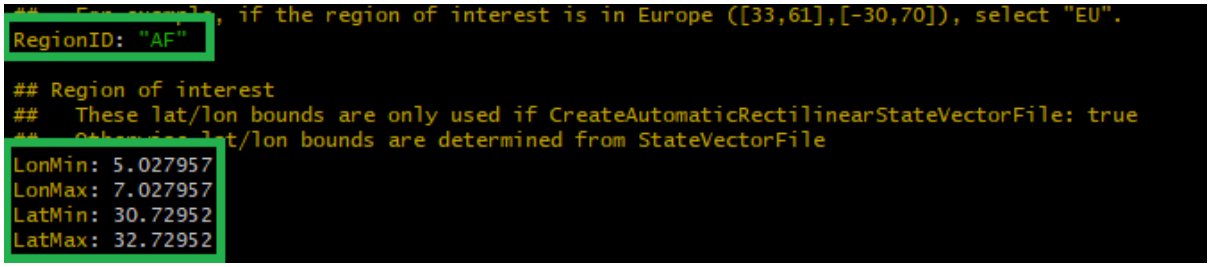

Figure 33: Analysis bounding box settings

Next there is a set of coordinates related to the Permian Basin default example that you don't want and should delete (fig.34)

Figure: 34: Point source information for Permian Basin example to be deleted.

Still further down is the IMI preview section which is by default set to "true". You will change this to "false" when deciding to go ahead with the inversion (fig. 35).

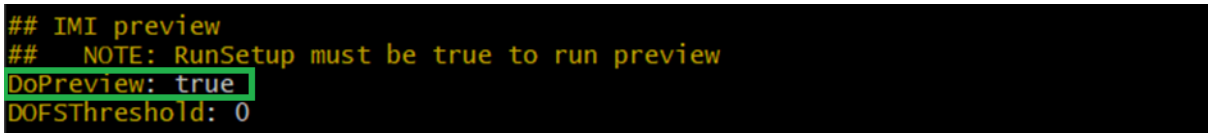

Figure 35: Config.yml DoPreview field set to true

Once you are satisfied with the settings, press F10 on your keyboard and using the arrow keys, navigate to "Exit", then press enter (fig. 36).

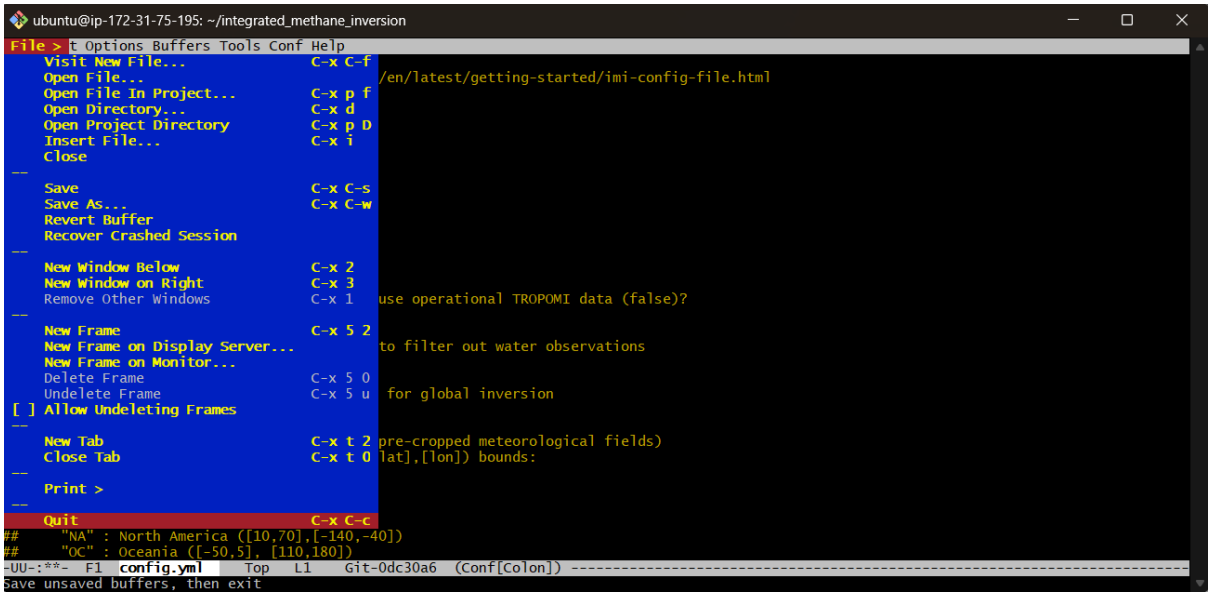

Figure 36: File menu in Config.yml

After exiting, run the following command to initiate the inversion preview:

```
sbatch run_imi.sh
```

and the following command to follow its progress:

tail --follow imi_output.log

Once the script has finished running it will look like figure 36 showing that the IMI preview for 1 week on the Permian Basin took around 15 minutes (fig. 37).

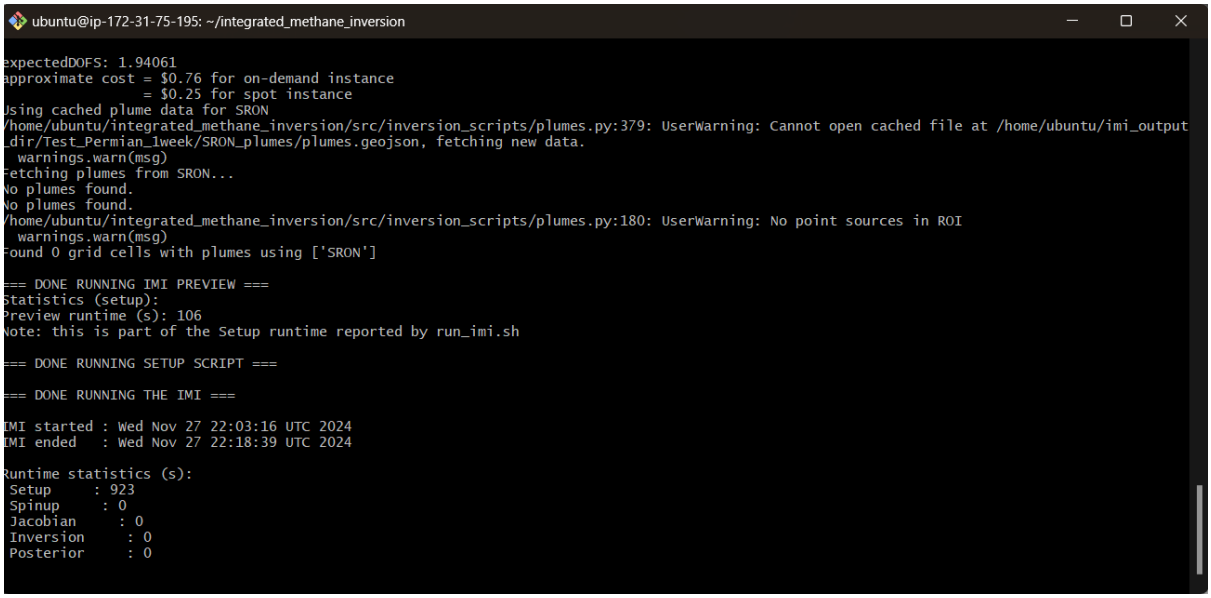

Figure 37: completed IMI Preview process.

2.3.1 Viewing the IMI Preview data

Open a new Git Bash terminal and log in as before (fig. 38)

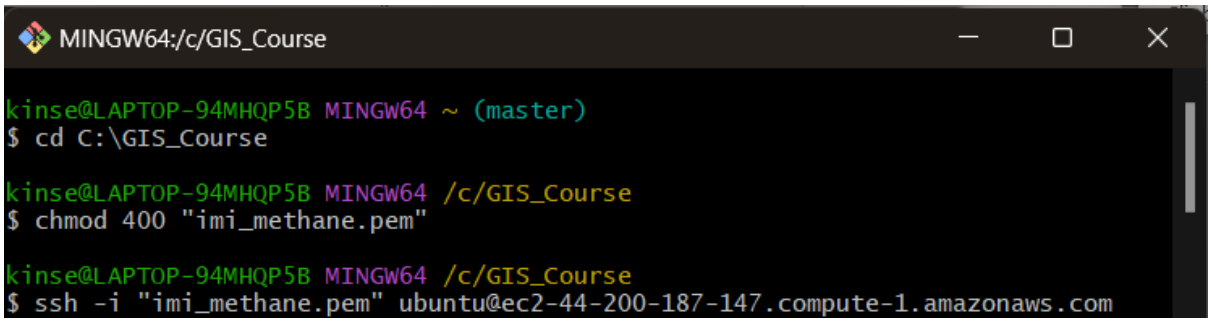

Figure 38: Login to instance

Once you have logged into the instance, enter the code:

jupyter notebook --no-browser --port 8080

This starts a Jupyter server on port 8080 and will print out two URLs with an authentication token. This as shown in figure 39.

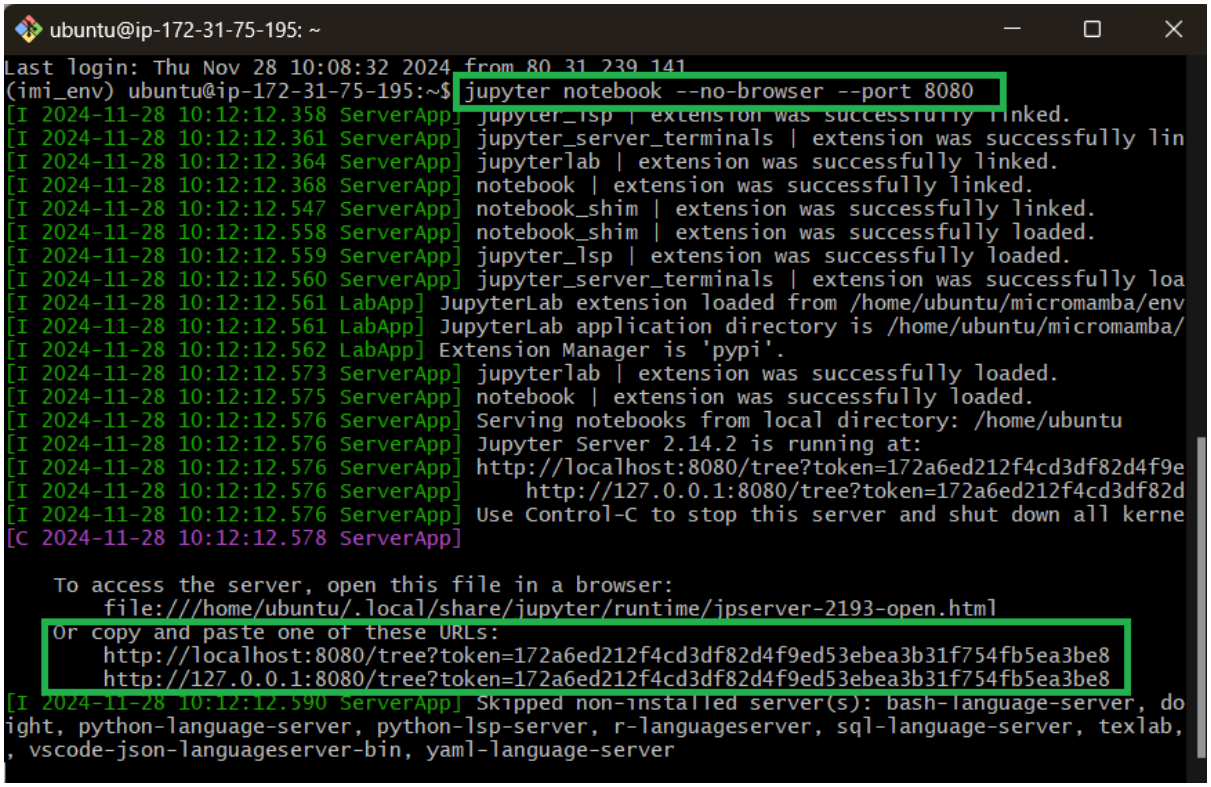

Figure 39: Jupyter server port on 8080 and URLs with authentication tokens.

Before you use these links you need to open an ssh tunnel from the ec2 instance to your local computer via port 8080. Open a new Git Bash terminal. Instead of logging into the instance as usual, you are going to use the following command to create the tunnel, changing the private key path to where your private key is stored on your local machine, and the host name to the same as the one shown in figure 40 on your connect to instance page.

Format: ssh -NL 8080:localhost:8080 -i /path/to/private_key ubuntu@<host-name>

Example: ssh -NL 8080:localhost:8080 -i "C:\GIS_Course\imi_methane.pem" ubuntu@ec2-3-238-122-254.compute-1.amazonaws.com

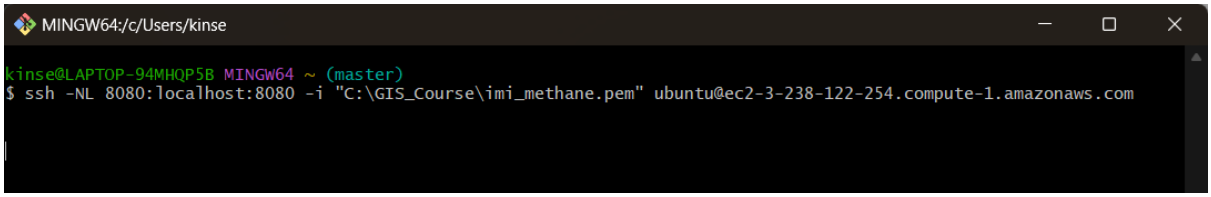

Figure 40: ssh tunnel command

Once you do this, nothing will seem to have happened in this Git Bash window but in the other window, commands will run to create the tunnel. You can now use one of the two URLs provided in figure 39 to access and view the IMI preview data (fig. 41).

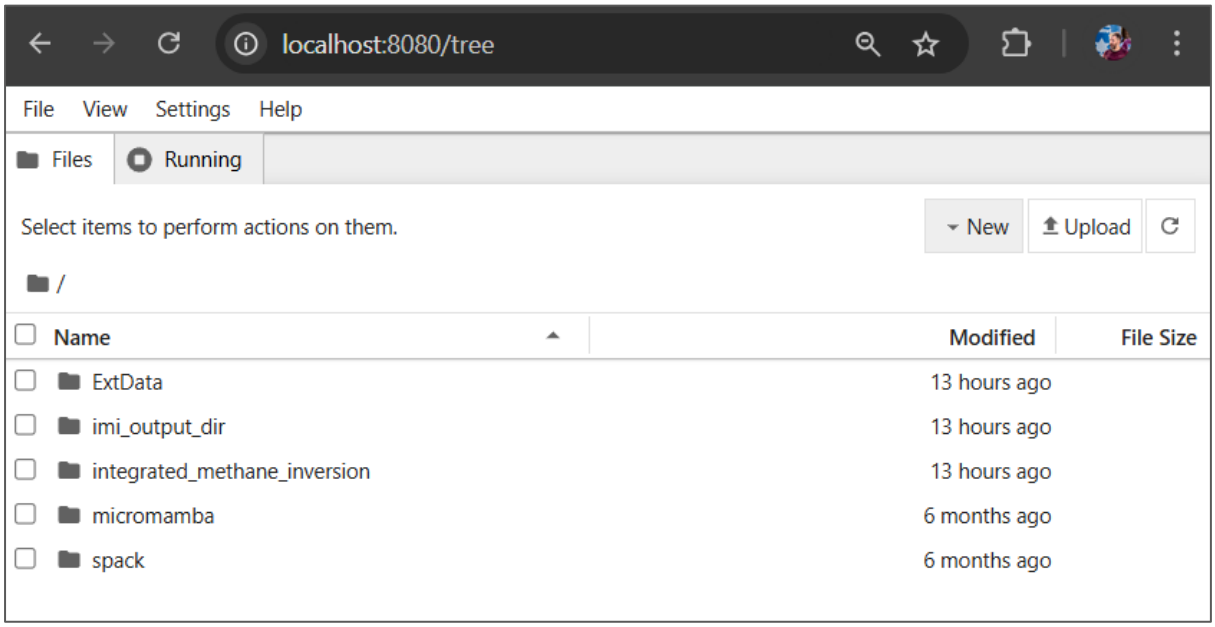

Figure 41: file directory of the IMI Preview

Once you have access to the file directory shown above, navigate to:

/imi_output_dir/(Your RunName)/preview/

From there you can download and view the IMI preview data (fig. 42)

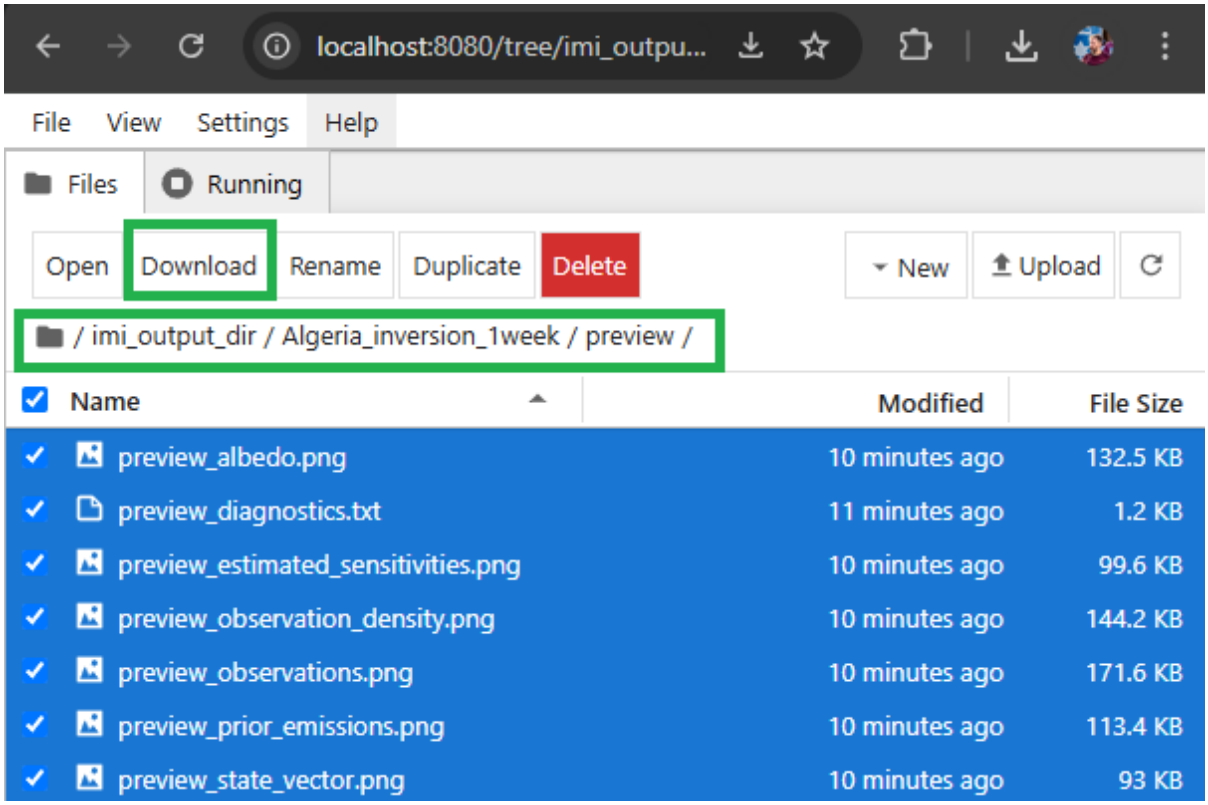

Figure 42: Location of IMI Preview data and download button

The IMI Preview will take approximately 15 minutes for a study size of 10,000km^2 of 1 week duration, with the total time scaling linearly.

2.3.2 Understanding the IMI Preview data

The IMI preview contains one .txt file and six maps, four of which are of interest. We will go through each of these in turn and explain which can help decide if you should proceed with the inversion. The first two maps to examine are the visualizations of TROPOMI whole atmosphere data (preview_observations.png) and the official "prior" emission estimates (preview_prior_emissions.png). The spatial distribution between these two maps should become more corelated, the longer the period studied, as atmospheric concentrations average out across emission sources. A poor spatial correlation over longer periods may suggest that the "prior" inventory data does not accurately reflect the actual emissions (Fig. 43).

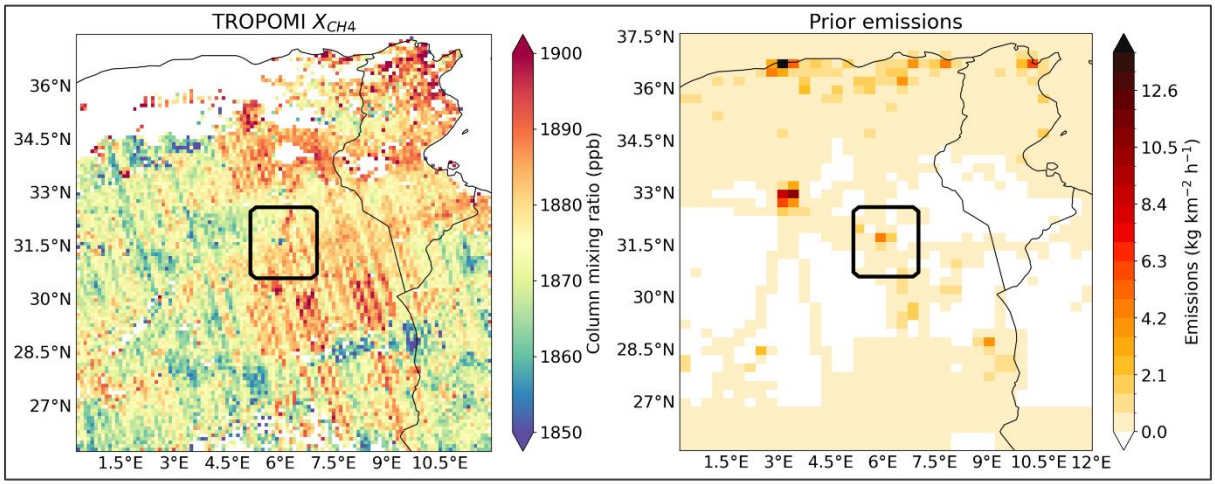

Figure 43: Mean TROPOMI Data for 1 week (left) and mapped "prior" inventory emissions (right)

The next map is of the observation density (preview_observation_density.png). In this example, the preview represents one week in May 2018, so the maximum observation density expected is 7 as Sentinel-5P has a daily return period (fig. 44). If there is significant cloud cover during the selected period, low observational density will be evident. In such cases, you may decide not to proceed with the inversion due to a lack of data.

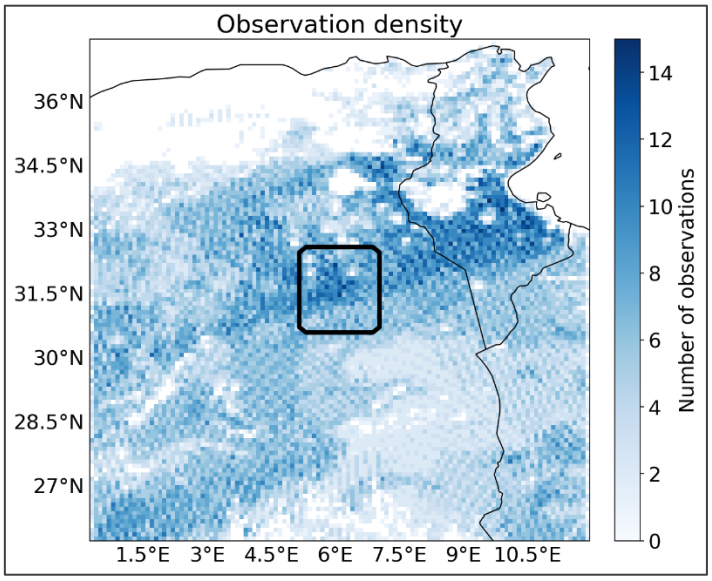

Figure 44: Observation density for 1 week

TROPOMI's measurements of surface albedo in the SWIR spectrum are visualized in the map labelled "preview_albedo.png" (fig. 45). Albedo is a critical factor in measuring gases like methane (CH_4) because

variations in surface reflectivity can lead to inaccurate readings. The IMI method accounts for these differences, but areas with very low albedo may overestimate emissions, while areas with high albedo may underestimate them (Barré et al., 2021). This map does not say much about if one should go ahead with an inversion or not, but examining it, users can identify potential false readings from the final IMI, particularly if the spatial patterns of albedo and methane retrievals appear similar.

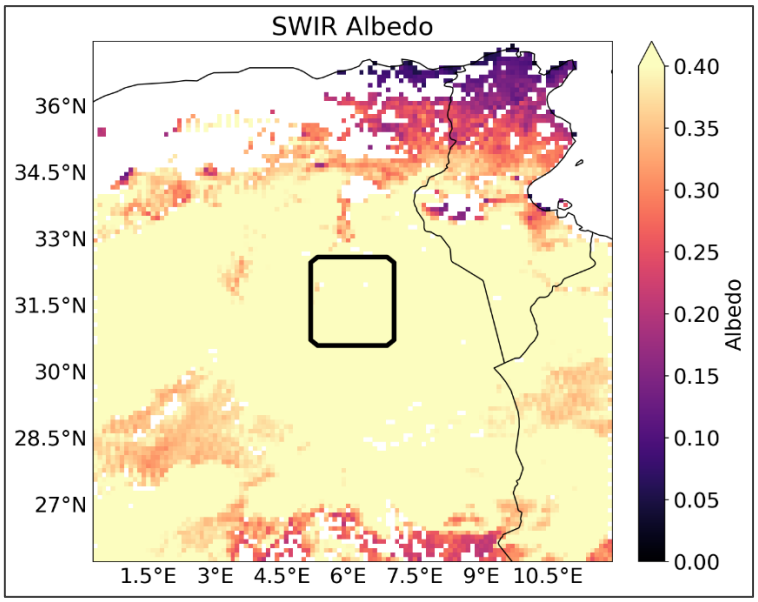

Figure 45: TROPOMI map of SWIR Albedo

The file "preview_diagnostics.txt" provides an estimate of the inversion's cost (shown in Figure 46) and a value called "expectedDOFS". DOFS, or Degrees of Freedom for Signal, measures how much the TROPOMI observations contribute to the final methane emission estimate compared to prior official inventory data. The DOFS value is influenced by the observation period; longer observation periods tend to increase DOFS, leading to more accurate results. A DOFS value of 1 is the minimum required to get meaningful emissions values, and higher values are preferred (Varon, et al. 2022). Shen et al. (2022) found that a DOFS greater than 2 was needed to reliably estimate emissions for oil and gas basins. If your DOFS is too low, you can extend the observation period and run the IMI preview again to improve the result.

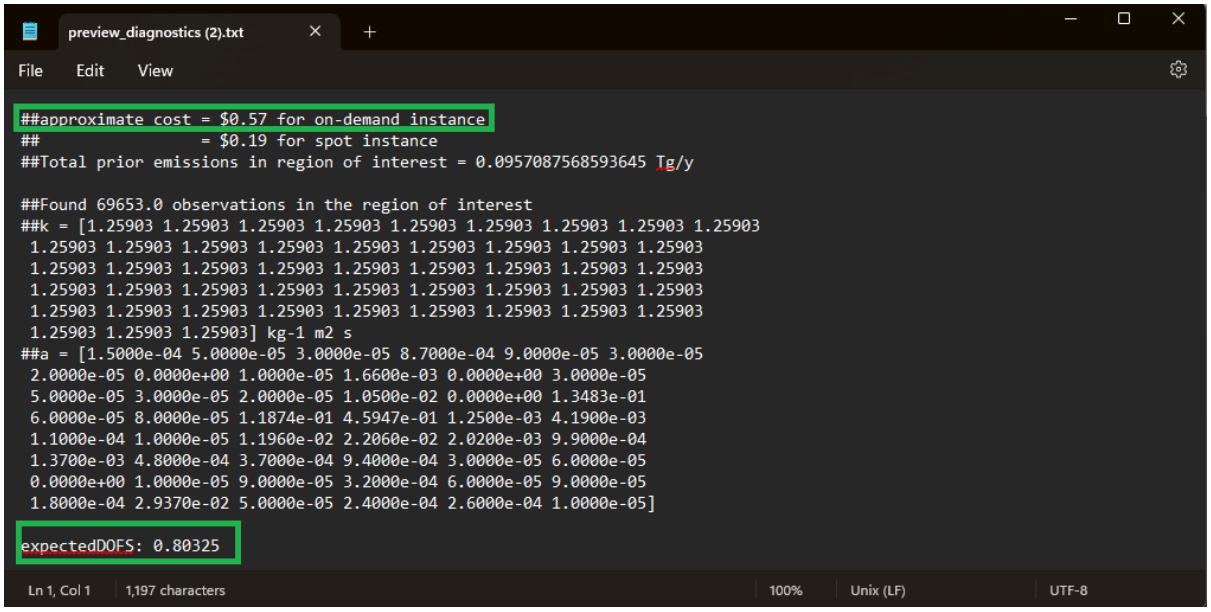

Figure 46: Preview_diagnostics.txt with estimated cost and DOES fields highlighted.

2.4 Configuring and running the IMI.

If the IMI preview looks correct and the cost is within your budget, you will now proceed with running the IMI. Open a new Git Bash terminal and connect to your instance as before (fig. 47).

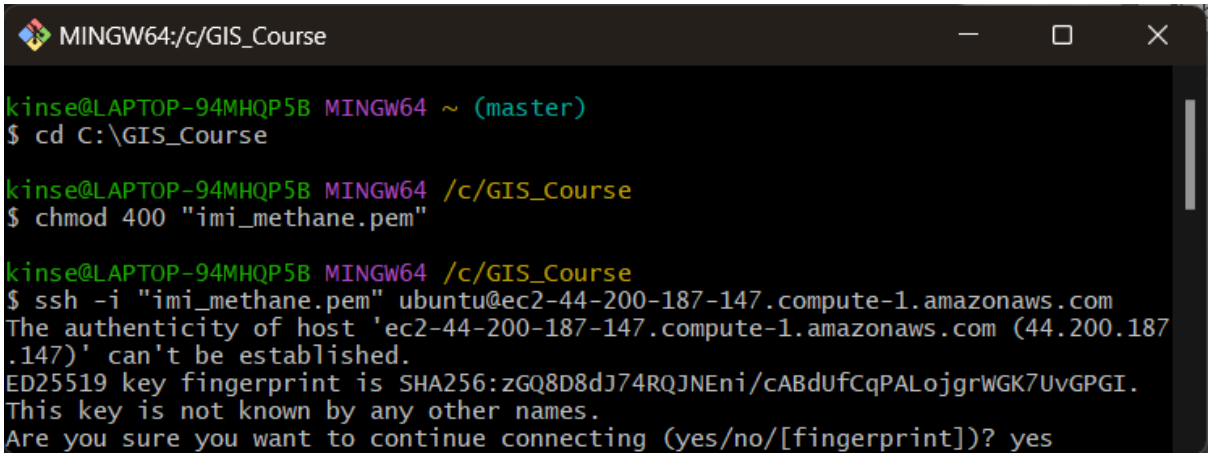

Figure 47: Connecting to instance.

In the Git Bash terminal, navigate to the IMI setup directory by typing:

cd ~/integrated_methane_inversion

Next open the inversion configuration file by typing the following:

emacs config.yml

The following screen should appear after a pause of a few moments (fig. 48)

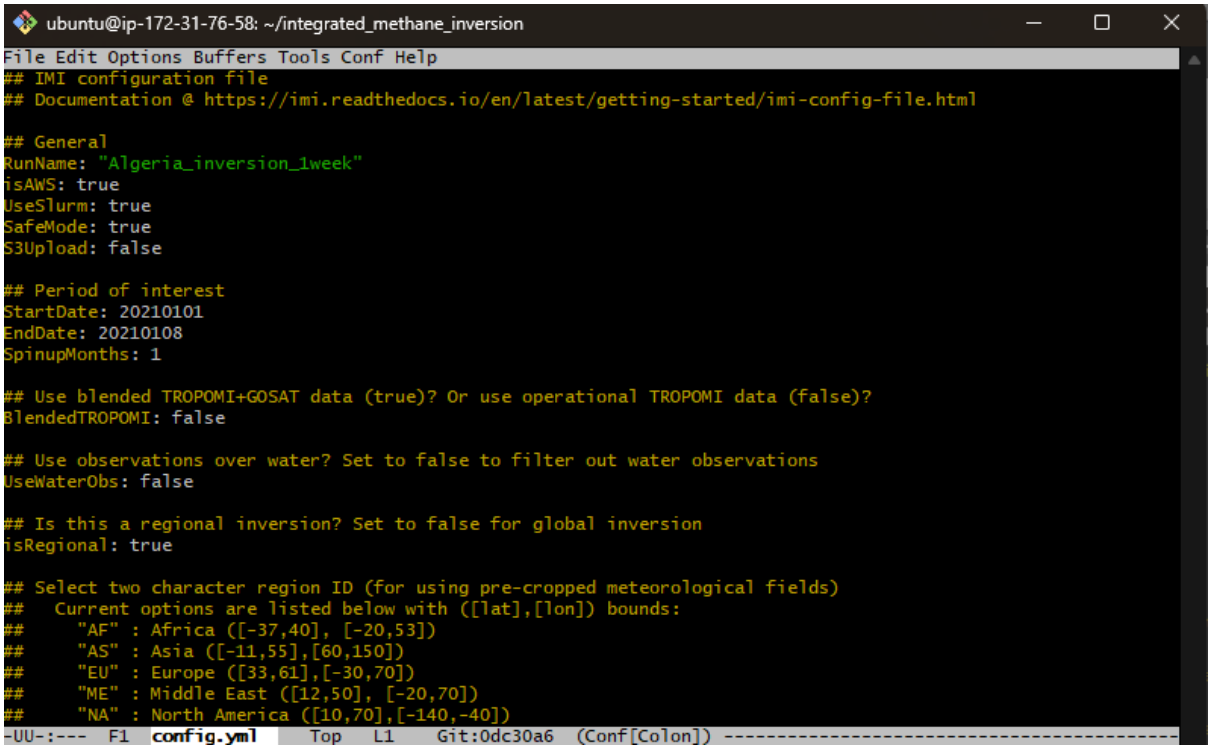

Figure 48: Config.yml, first section.

Before you changed the following fields to match the area and time you were interested in.

- StartDate
- EndDate
- LongMin
- LongMax
- LatMin
- LatMax

Now you are going to set the following fields to the values below to run the IMI instead of the IMI preview.

Setup modules

Turn on/off different steps in setting up the inversion

- RunSetup: true
- SetupTemplateRundir: false
- SetupSpinupRun: true
- SetupJacobianRuns: true
- SetupInversion: true
- SetupPosteriorRun: true

Run modules

Turn on/off different steps in performing the inversion

- DoHemcoPriorEmis: false
- DoSpinup: true
- DoJacobian: true

- ReDoJacobian: true
- DoInversion: true
- DoPosterior: true

IMI preview

NOTE: RunSetup must be true to run preview

- DoPreview: false

Once you have adjusted those settings, press F10 and save the changes (fig. 49), then press F10 again and Quit to return to the console.

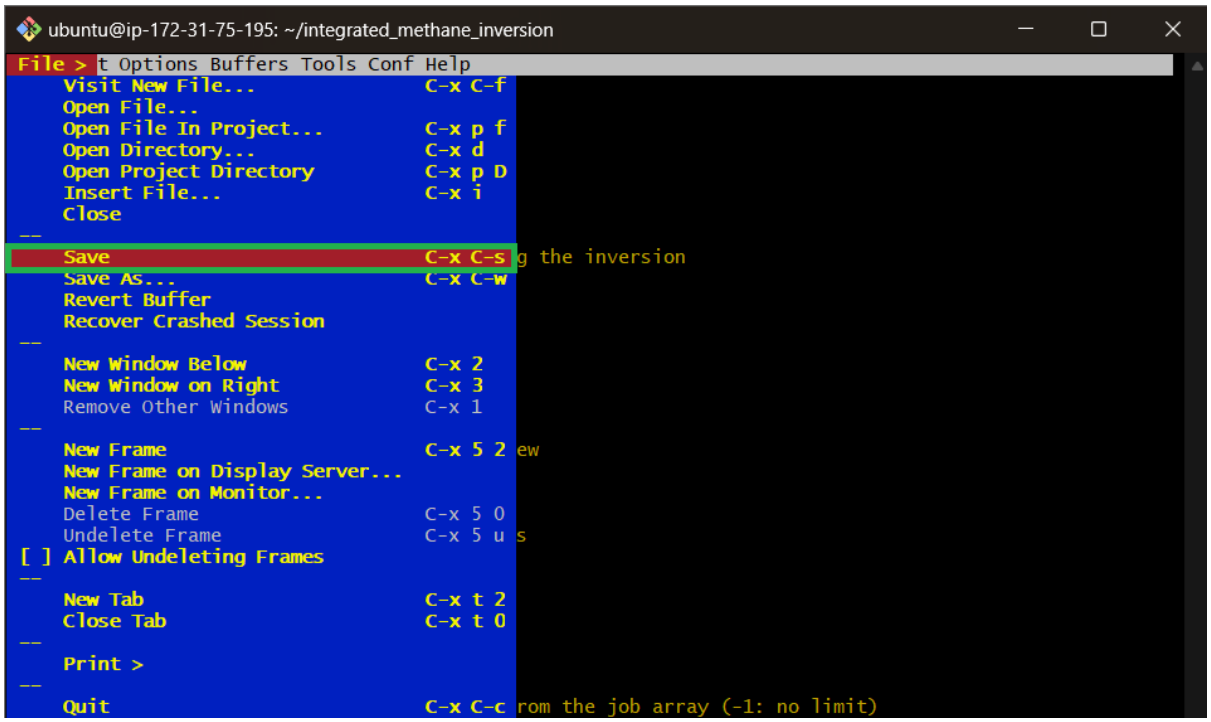

Figure 49: Location of the save button in the config.yml screen.

In the Git Bash terminal issue this command to run the IMI:

sbatch run_imi.sh

As before you can follow its progress by using the following command:

tail --follow imi_output.log

As shown in figure 50.

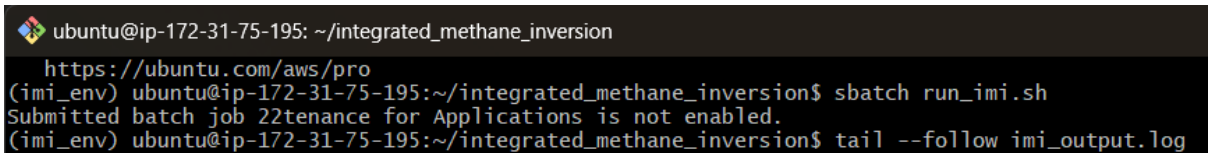

Figure 50: Commands running IMI and following progress.

At this point you can disconnect from your instance. The IMI will take approximately 2 and a half hours to run for a study size of 10,000km^2 of 1 week duration, with the total time scaling linearly.

2.4.1 Accessing the IMI data

Open a new Git Bash terminal and log in to your instance (fig. 51)

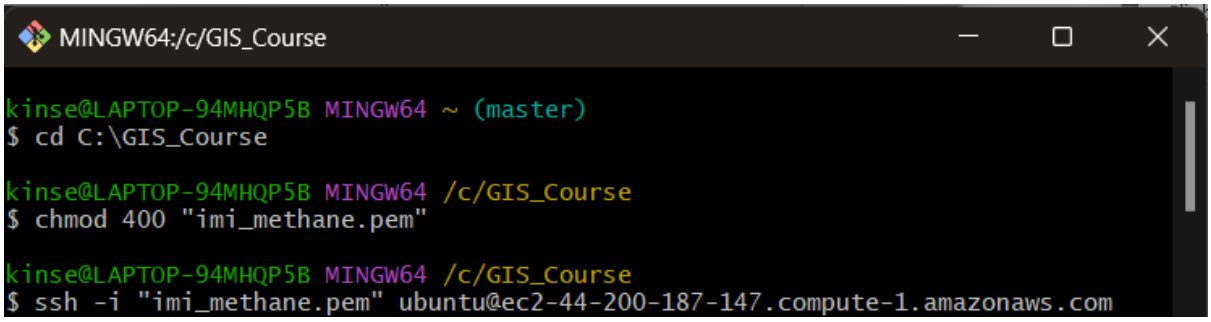

Figure 51: Connecting to instance.

Once you have logged into the instance, enter the code:

jupyter notebook --no-browser --port 8080

This starts a Jupyter server on port 8080 and will print out two URLs with an authentication token. This as shown in figure 52.

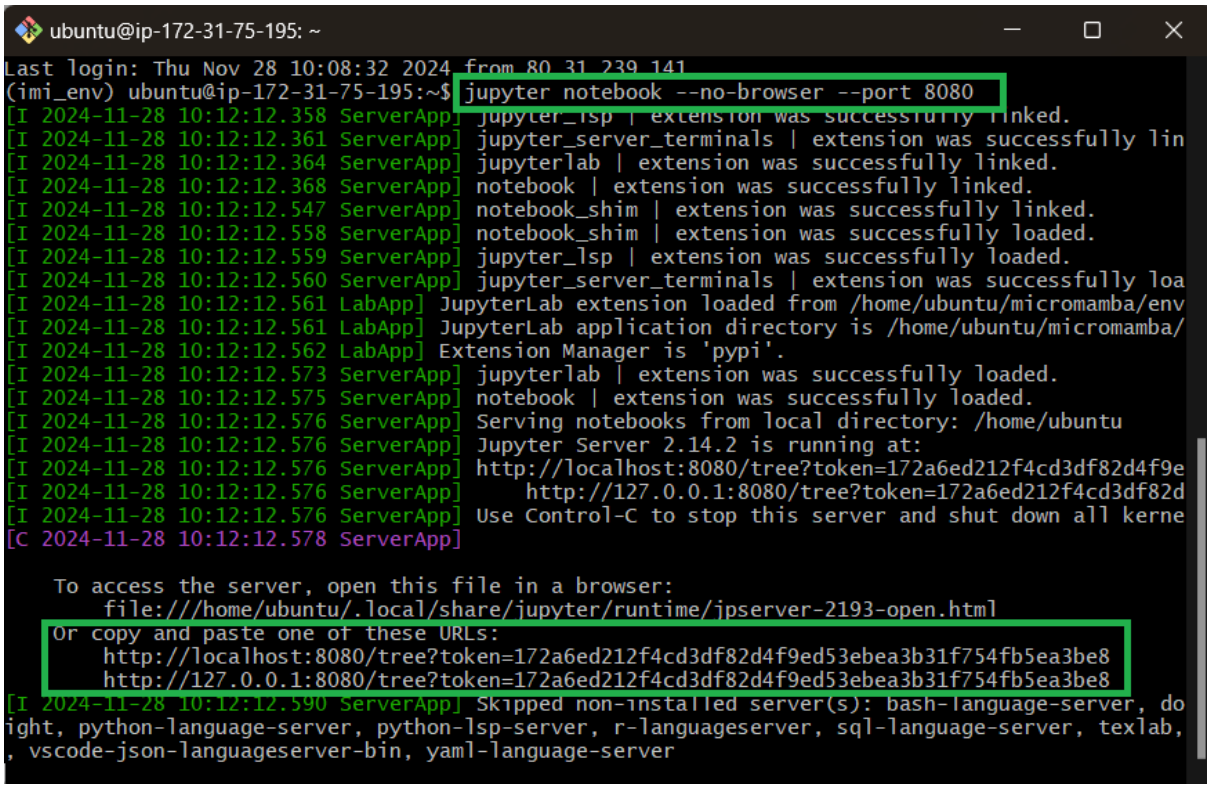

Figure 52: Jupyter server port on 8080 and URLs with authentication tokens.

Before you use these links you need to open an ssh tunnel from the ec2 instance to your local computer via port 8080. Open a new Git Bash terminal. Instead of logging into the instance as usual, you are going to use the following command to create the tunnel, changing the private key path to where your private key is stored on your local machine, and the host name to the same as the one shown in figure 27 on your connect to instance page (fig. 53).

Format: ssh -NL 8080:localhost:8080 -i /path/to/private_key ubuntu@<host-name>

Example: ssh -NL 8080:localhost:8080 -i "C:\GIS_Course\imi_methane.pem" ubuntu@ec2-3-238-122-254.compute-1.amazonaws.com

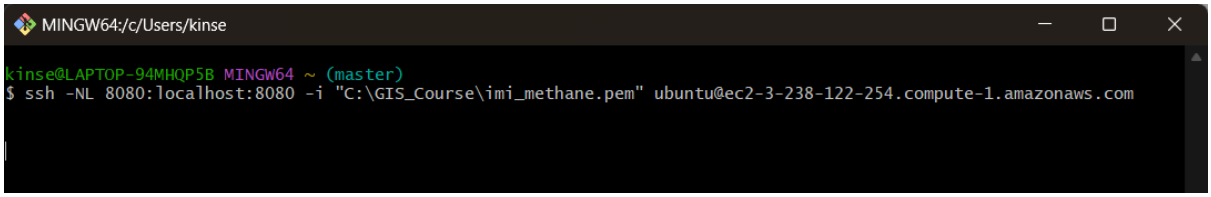

Figure 53: ssh tunnel connection

Once you do this, nothing will seem to have happened in this Git Bash window but there but in the background the tunnel will have been created. You can now use one of the two URLs provided in figure 61 to access and view the IMI preview data (fig. 54).

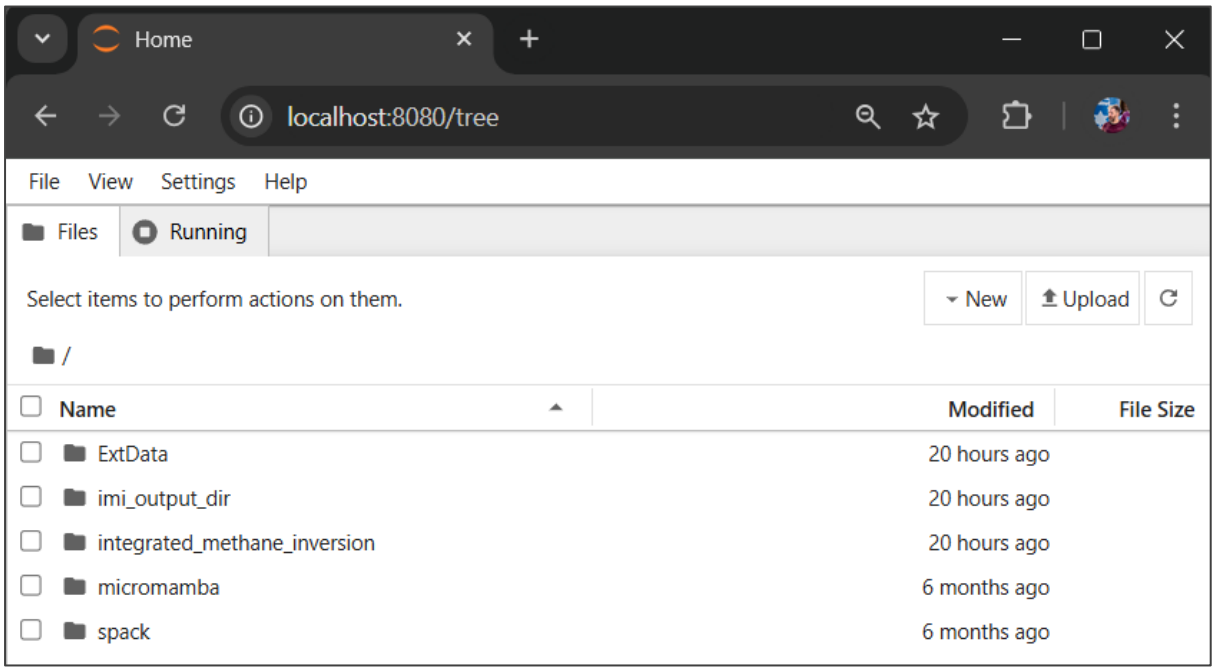

Figure 54: IMI file directory

Navigate to:

/imi_output_dir/Test_Permian_1week/inversion/

and look for "visualization_notebook.ipynb". Open this and a Jupyter notebook will appear. In this notebook, select from the menu "Run" and then "Run all cells" (fig. 55).

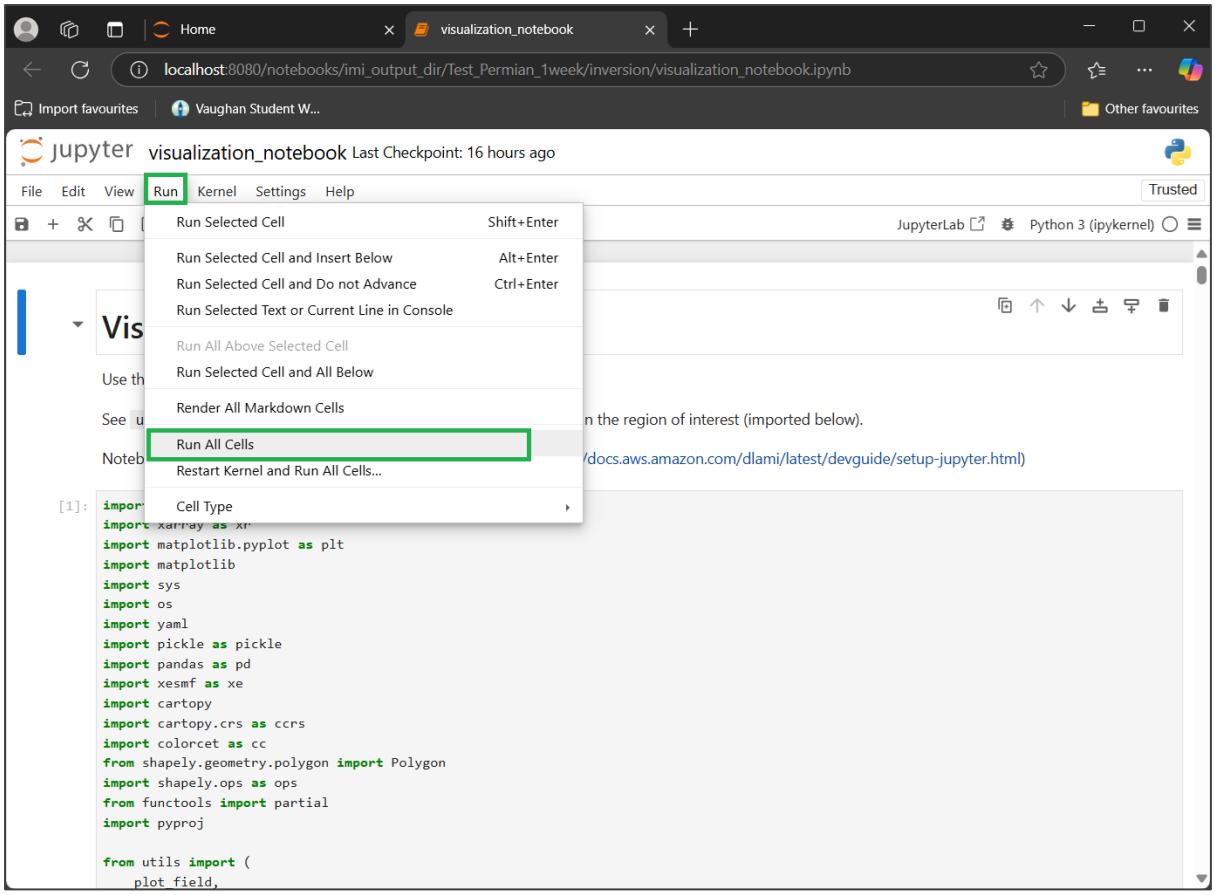

Figure 55: Run all cells command

2.4.2 Interpreting the IMI data

Once this is complete, scroll down to view the data. The first point of interest is code box 8 which shows the previous estimate for emissions based on official inventories (Prior) and the measured emissions based on Sentinel-5P and IMI (Posterior). As can be seen in figure 56, the official "prior" estimate is 0.079 Teragrams per year (Tg/y) below the IMI measured figure, which is 79,000 metric tonnes.

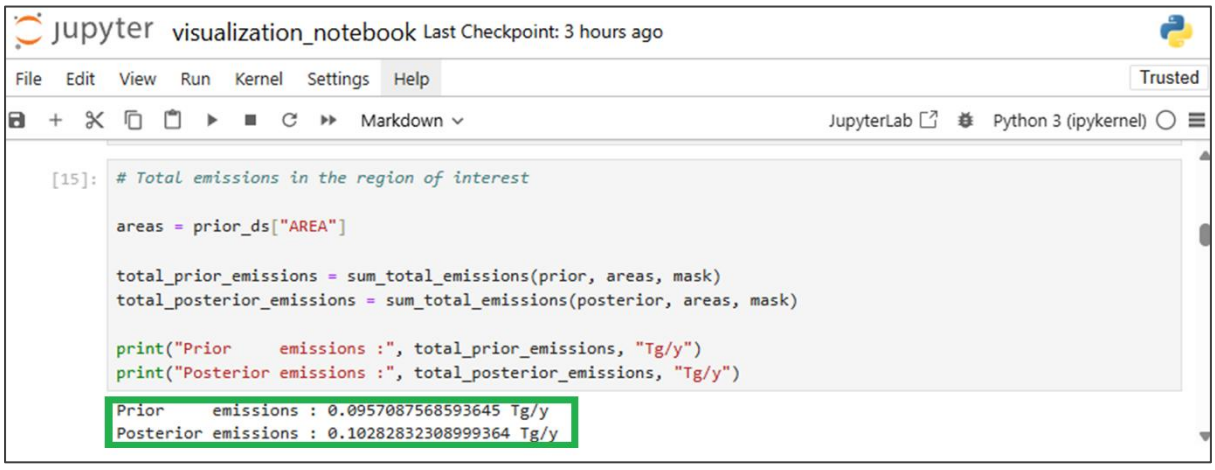

Figure 56: Official "prior" inventory emissions and Measured "posterior" emission totals.

Code box 9 further visualises this and shows the official "prior" estimates on a map, whereas code box 10 shows the mapped IMI measured "posterior" emissions (fig. 57).

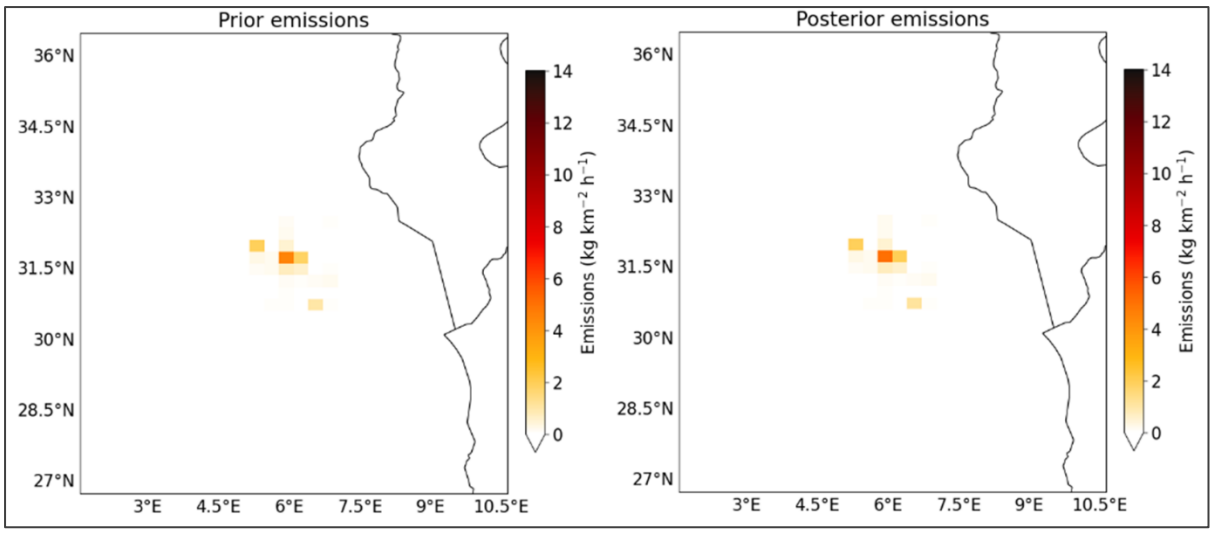

Figure 57: Mapped official "prior" inventory emissions and mapped "posterior" emissions

Code box 11 shows the sectoral emissions. This is determined by applying the ratios given by the official "prior" estimates to the IMI results (fig. 58).

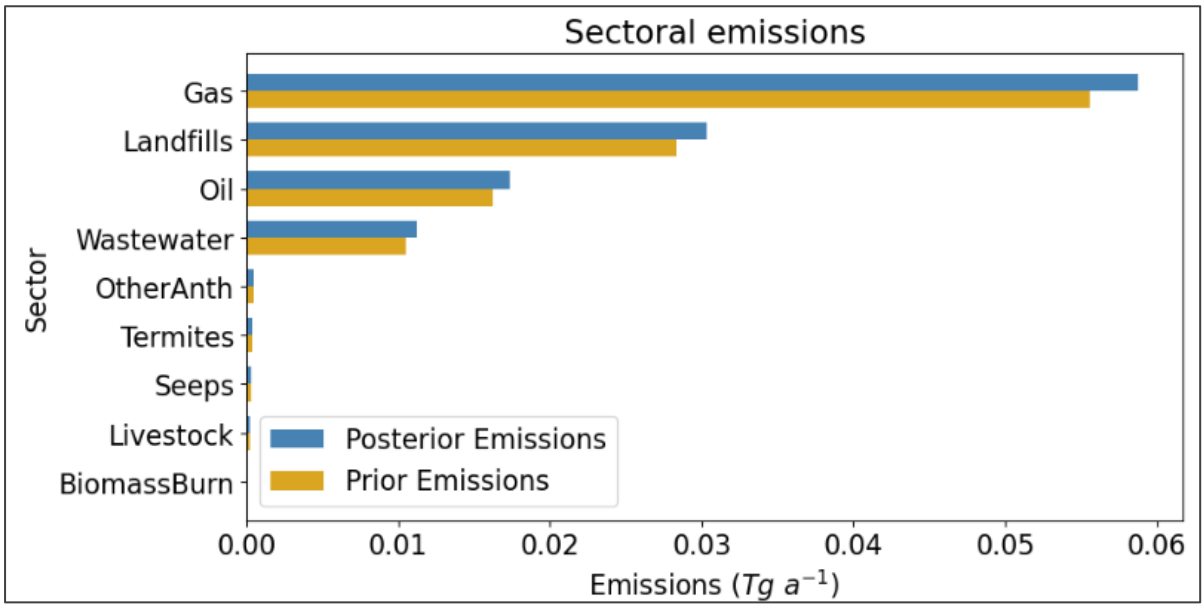

Figure 58: Emissions as divided by sector

If you wish to download any of the figures from the visualisation notebook, you can do so by navigating to:

/imi_output_dir/(Your RunName)/inversion/output/

Selecting the tick boxes for the figures and then clicking the download button (fig. 59)

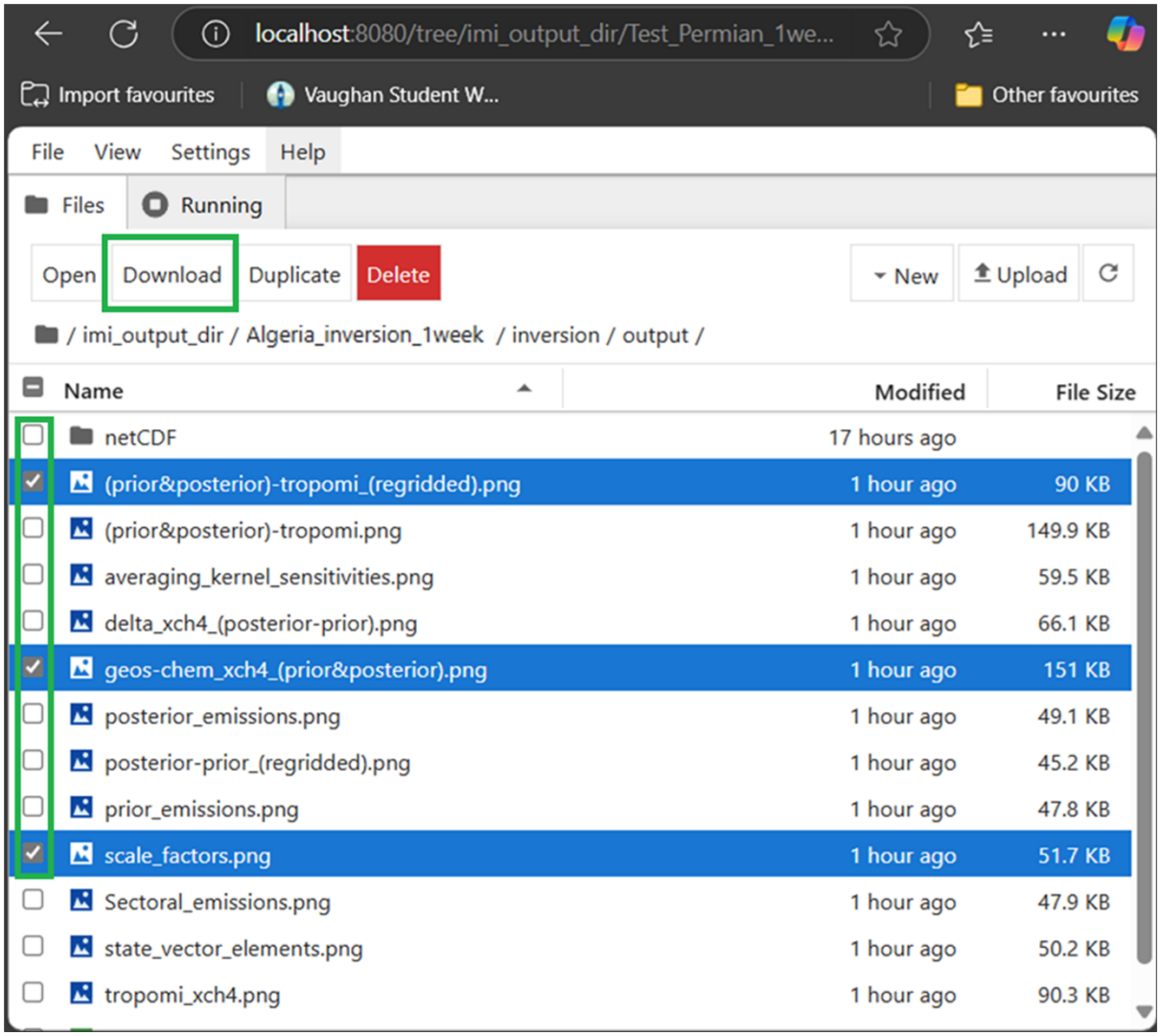

Figure 59:Location of check boxes and downloads.

2.5 Oil and gas field bounding parameters

101 fields were mapped in this project's survey. This was done by georeferencing the map in Abada and Bouharkat (2018) and then manually surveying the Imagery basemap in ArcGIS Pro. It is therefore, not perfect, and should ideally be replaced by official data when possible. On the following page, figure 60 shows a map of numbered fields which each correspond to a particular bounding box of around 10,000km around that field. Once you have found the field that you are interested in monitoring, please consult table 1 for its specific bounding coordinates, which are taken as 1 degree east, west, north and south of its centroid.

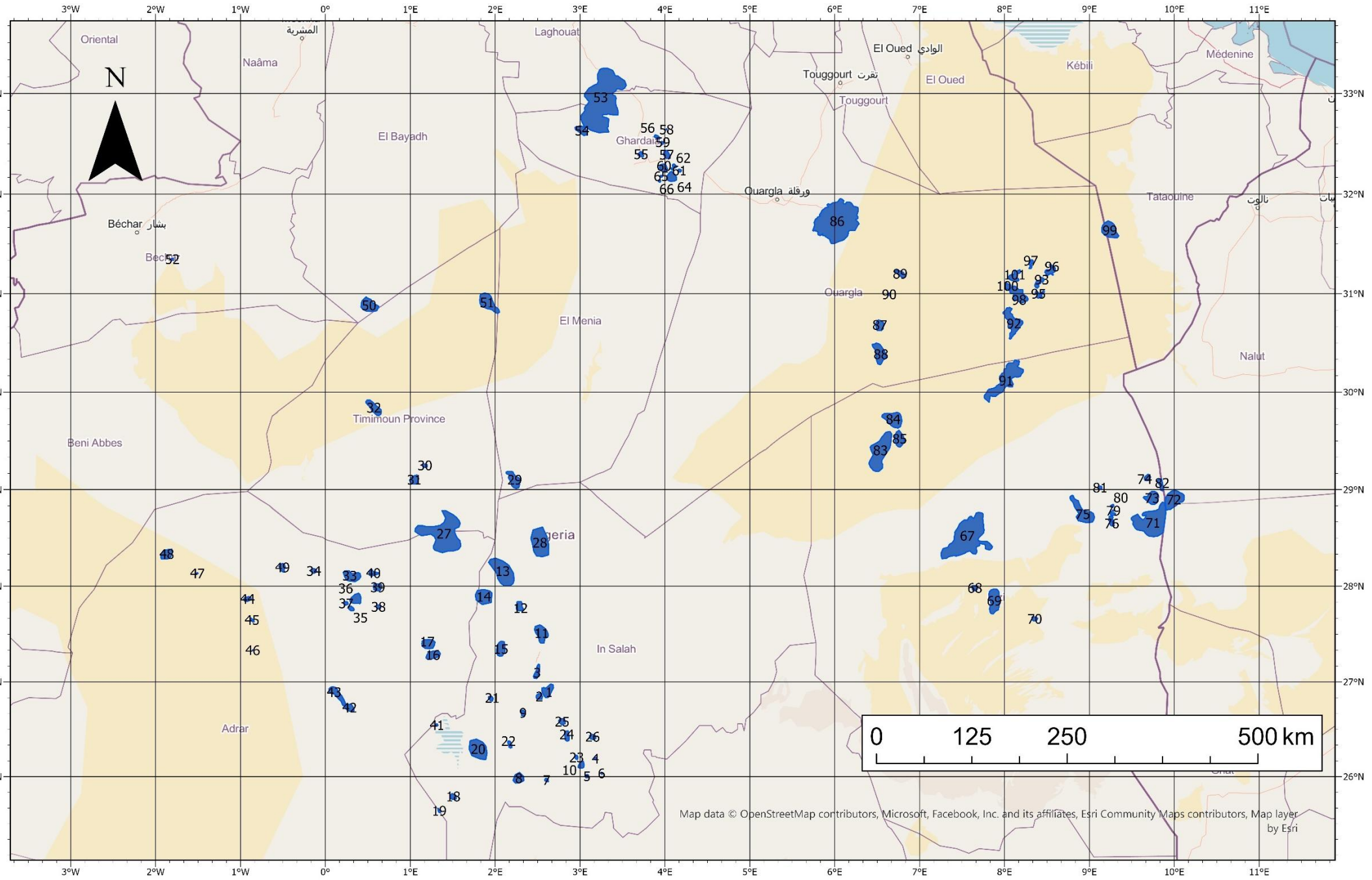

Figure 60: Numbered oil and gas field map

Table 1: Boundings for mapped Algerian fields

Nº	Field Name	LongMin	LongMax	LatMin	LatMax
1	Gour Mahmoud	1.618772	3.618772	25.90473	27.90473
2	Hassi Hassine	1.520698	3.520698	25.84444	27.84444
3	In Sallah	1.49459	3.49459	26.10151	28.10151
4	Mahbes Guenatir	2.182007	4.182007	25.19104	27.19104
5	Djebel Thara	2.082693	4.082693	25.00474	27.00474
6	Unknown	2.254464	4.254464	25.03616	27.03616
7	Krebb Ed Douro	1.606319	3.606319	24.96279	26.96279
8	Tibardine	1.276927	3.276927	24.98251	26.98251
9	Oued Djaret	1.329972	3.329972	25.67153	27.67153
10	Djebel Mouahdrine	2.016066	4.016066	25.12113	27.12113
11	Hassi Moumene	1.544895	3.544895	26.50759	28.50759
12	Garet El Befinat	1.286967	3.286967	26.79062	28.79062
13	Reg	1.090235	3.090235	27.14886	29.14886
14	Bouteraa	0.867103	2.867103	26.89035	28.89035
15	Garet El Guef	1.067896	3.067896	26.34577	28.34577
16	Oued Talha	0.269069	2.269069	26.28196	28.28196
17	Hassi M'Sari	0.213089	2.213089	26.39746	28.39746
18	Adrar Morrat NE	0.507646	2.507646	24.78721	26.78721
19	Adrar Morrat SW	0.342025	2.342025	24.6368	26.6368
20	Bahar El Hammar	0.799932	2.799932	25.28829	27.28829
21	Tit	0.949045	2.949045	25.82488	27.82488
22	Tirechoumine	1.173883	3.173883	25.34059	27.34059
23	Anasmit	1.958895	3.958895	25.20431	27.20431
24	In Bazzene	1.845299	3.845299	25.43049	27.43049
25	In Bazzene N	1.789646	3.789646	25.58308	27.58308
26	Djebel Belda	2.149968	4.149968	25.41968	27.41968
27	Hassi Barouda	0.39071	2.39071	27.54333	29.54333
28	Teguentour	1.529539	3.529539	27.45318	29.45318
29	Krechba	1.22075	3.22075	28.10089	30.10089
30	Erg Choueref	0.172232	2.172232	28.24736	30.24736
31	Bejouen	0.048506	2.048506	28.10105	30.10105
32	Ramedj	-0.4258	1.574196	28.8392	30.8392
33	Oued Zine	-0.68848	1.311522	27.10283	29.10283
34	Hassi Sbaa	-1.13753	0.862465	27.15955	29.15955
35	Hassi Llatoou	-0.69677	1.303225	26.76801	28.76801
36	Hassi Llatoou NE	-0.63814	1.361858	26.86594	28.86594
37	Hassi Llatoou Cambrien	-0.75645	1.243551	26.82124	28.82124
38	Qued Tourhar	-0.3727	1.627301	26.7855	28.7855
39	Azzene	-0.386	1.614004	26.9898	28.9898
40	Foukroun	-0.43969	1.560307	27.13475	29.13475
41	Mekerrane N	0.312063	2.312063	25.54544	27.54544
42	Azrafil SE	-0.71332	1.286684	25.72597	27.72597
43	Reggane	-0.85935	1.140647	25.86477	27.86477
44	Kahal Tabelbala N	-1.91849	0.081513	26.866	28.866
45	Djebel Heirane Kahal	-1.86152	0.138483	26.64626	28.64626
46	Djebel Heirane N	-1.8551	0.144902	26.33112	28.33112
47	Feidj El Had	-2.50793	-0.50793	27.13334	29.13334
48	Hassi M'Dakane	-2.869	-0.869	27.33109	29.33109
49	Touat / Decheira	-1.50686	0.49314	27.19443	29.19443
50	Hassi Tidjerane	-0.48716	1.512844	29.8808	31.8808

51	Hassi Ba Hammou	0.916395	2.916395	29.90914	31.90914
52	Meharez	-2.80463	-0.80463	30.34845	32.34845
53	Hassi R'Mel	2.257761	4.257761	31.92535	33.92535
54	Hassi R'Mel S	2.019565	4.019565	31.63556	33.63556
55	Macouda	2.715795	4.715795	31.3953	33.3953
56	Djorf	2.908041	4.908041	31.56769	33.56769
57	Oued Noumer	3.022115	5.022115	31.39291	33.39291
58	Unknown	3.018276	5.018276	31.64427	33.64427
59	Oued Noumer	2.968834	4.968834	31.5101	33.5101
60	Oued Noumer	2.982743	4.982743	31.26112	33.26112
61	Unknown	3.171823	5.171823	31.23443	33.23443
62	Unknown	3.110875	5.110875	31.28077	33.28077
63	Unknown	3.085245	5.085245	31.25313	33.25313
64	Ait Kheir	3.078018	5.078018	31.16998	33.16998
65	Unknown	2.956173	4.956173	31.17799	33.17799
66	Unknown	2.936542	4.936542	31.13219	33.13219
67	Fouye Tabankort	6.559587	8.559587	27.50264	29.50264
68	Adouhoum	6.650705	8.650705	26.9808	28.9808
69	Mezoratine	6.878882	8.878882	26.84833	28.84833
70	Remal	7.354091	9.354091	26.65971	28.65971
71	Alrar	8.732503	10.7325	27.66768	29.66768
72	Alar (Lybia)	8.993689	10.99369	27.89798	29.89798
73	Stah	8.741478	10.74148	27.91423	29.91423
74	Mereksene	8.683642	10.68364	28.12191	30.12191
75	Ohanet	7.918675	9.918675	27.76401	29.76401
76	Dimera	8.26935	10.26935	27.64778	29.64778
77	Dimera	8.254425	10.25443	27.70002	29.70002
78	Dimera	8.271056	10.27106	27.75194	29.75194
79	Dimera	8.282382	10.28238	27.78212	29.78212
80	Dimera	8.269205	10.2692	27.83156	29.83156
81	Dimera	8.128329	10.12833	28.02007	30.02007
82	Antar	8.84134	10.84134	28.05523	30.05523
83	Rhourde Adra S	5.535668	7.535668	28.38926	30.38926
84	Rhourde Nouss	5.692757	7.692757	28.72144	30.72144
85	Rhourde Adra	5.76142	7.76142	28.52019	30.52019
86	Hassi Messaoud	5.027957	7.027957	30.72952	32.72952
87	Nezia	5.529253	7.529253	29.6774	31.6774
88	Gassi Touil	5.528674	7.528674	29.39561	31.39561
89	Rhourde Akbar	5.777513	7.777513	30.19983	32.19983
90	Rhourde Sayah	5.641731	7.641731	29.99307	31.99307
91	El Merk	7.012159	9.012159	29.12799	31.12799
92	Ourhoud	7.0936	9.0936	29.71162	31.71162
93	Bir Sal Fatima	7.442603	9.442603	30.14124	32.14124
94	Bir Sal Fatima	7.388387	9.388387	30.09571	32.09571
95	Bir Sal Fatima	7.412735	9.412735	29.9916	31.9916
96	Bir Sal Fatima	7.54193	9.54193	30.22964	32.22964
97	Bir Sal Fatima	7.313152	9.313152	30.30008	32.30008
98	Qoubba N	7.165209	9.165209	29.98842	31.98842
99	El Borma	8.238376	10.23838	30.64019	32.64019
100	Hassi Berkine	7.035809	9.035809	30.07576	32.07576
101	Hassi Berkine	7.120797	9.120797	30.17533	32.17533

2.6. Shutting down the instance

Once you have run the inversion and collected the data you need, you must shut down the inversion to avoid any unnecessary costs. Open a browser and navigate to https://aws.amazon.com/, then login to your console using the user details you set up previously (fig. 61).

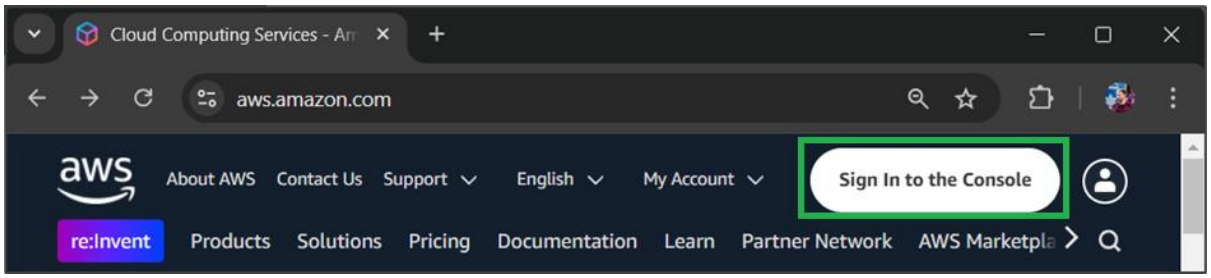

Figure 61: Sign into console button.

Once you have logged into your console, click on the EC2 Global view link (fig. 62).

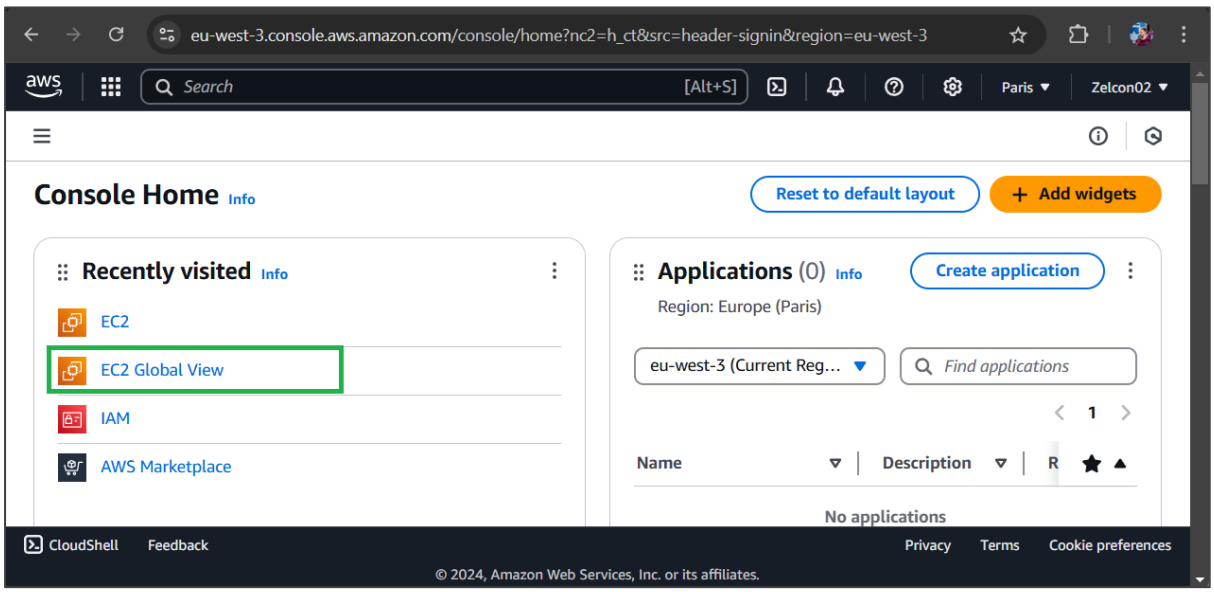

Figure 62: EC2 Global View button

You should see under “instances” 1 in 1 region (or however many instances you set up). Click on that link (fig. 63) and select “view all”.

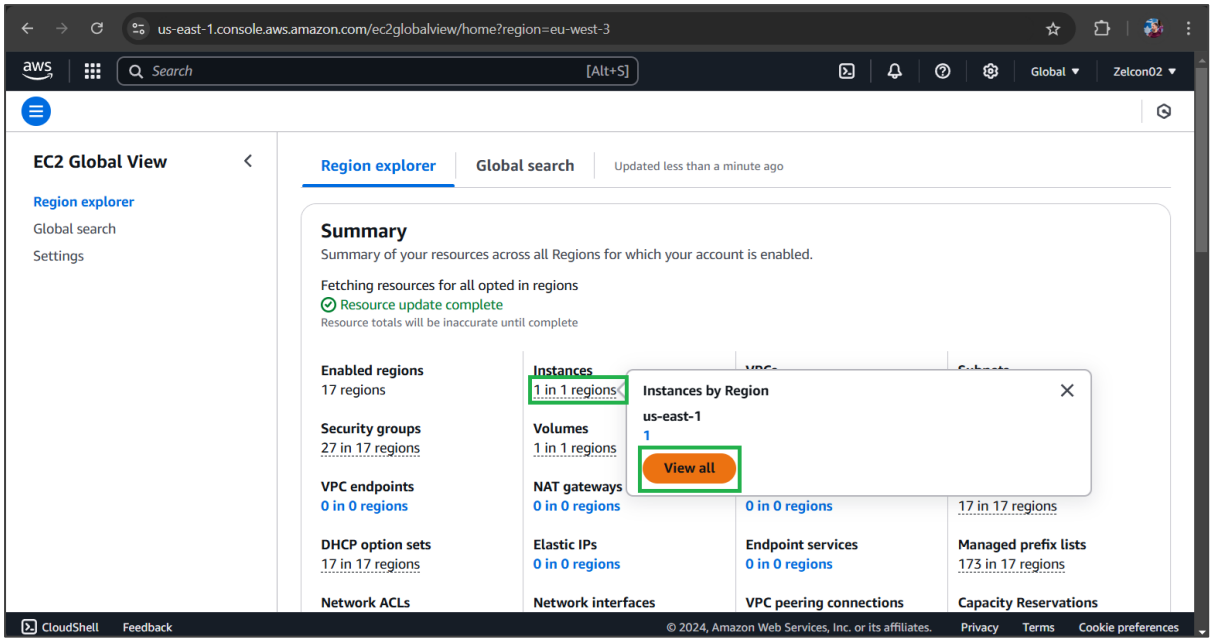

Figure 63: Instances field showing how many instances you have open.

Next click on the blue "resource ID" link (fig. 64)

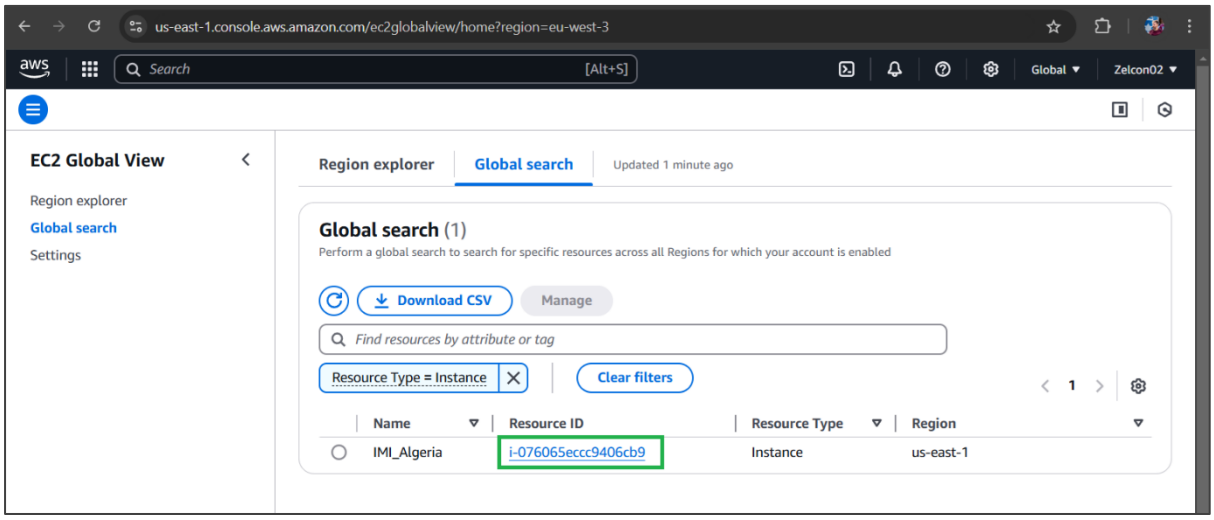

Figure 64: resource DI link location.

Finally open the "instance state" dropdown menu and select "stop instance" (fig. 65). Thereafter you should receive a message confirming the operation.

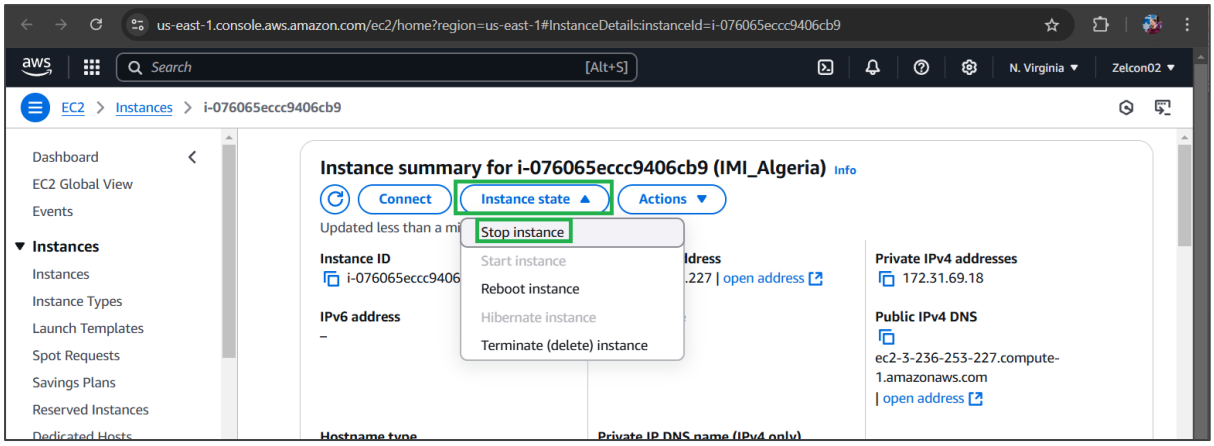

Figure 65: location of stop instance button.

2.7 Troubleshooting

If you are having problems running the IMI, please submit a ticket at the following Github page where the developers will attempt to help you out.

https://github.com/geoschem/integrated_methane_inversion/issues

3. Sentinel-2 Plume Finder

3.1 Methodology

To help users locate emission sources in a field, Sentinel 2's Multispectral Instrument with a 20m spatial resolution and return frequency of 3-5 days was employed.

Varon et al. (2021) demonstrated that methane columns from point sources can be measured by exploiting the SWIR-1 and SWIR-2 bands of Sentinel-2. The software uses Sentinel 2's L1C data accessed via the Copernicus OpenEO Application Program Interface (API) (Musial et al., 2024). The data is filtered to only use scenes with a maximum of 5% cloud cover. No emission and active emission scenes were chosen as two sequential passes and processed as in Varon et al. (2021), first using a MBSP method:

$$MBSP = \frac{B12 - B11}{B11}$$

Where: ***B12*** is the Sentinel-2 SWIR-2 band and ***B11*** is the Sentinel-2 SWIR-1 band.

Once the MBSP raster had been calculated, the following equation was then used to calculate the multi-band-multi-pass raster:

$$MBMP = ActiveMBSP - NoMBSP$$

Where **ActiveMBSP** is the multiband single pass for the active emission scene and **NoMBSP** is the multiband single pass for the no emission scene.

Figure 66 provides an overview of the main steps in processing.

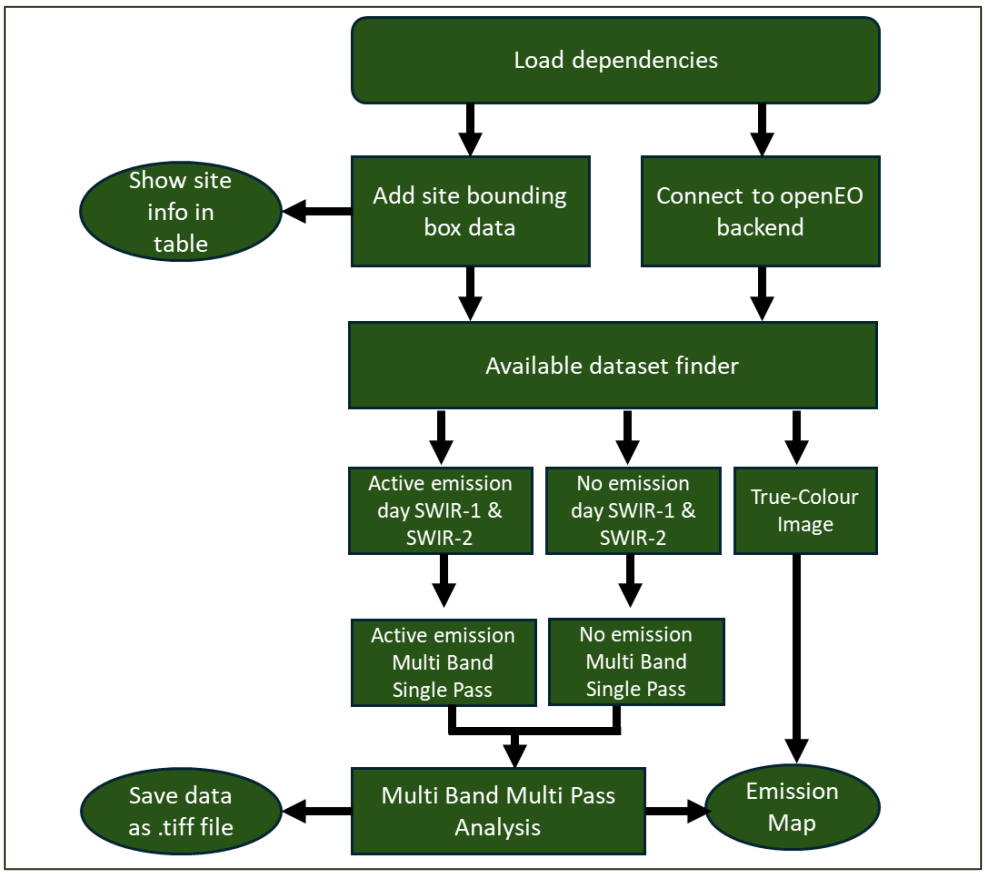

Figure 66: Flowchart of processes used for Sentinel 2 Plume Finder

3.2 Data and Dependencies

The datasets used by these tools are outlined in table 2.

Table 2: Datasets used by tools

Description	Data	Source	For use in tool(s)
Oil and Gas Field Bounding	Long/Lat .csv file.	Manual surveying of arial images of Algeria by author	S2-MBMP
Sentinel 2 (MSI) bands 2, 3, 4, 11 and 12	Raster 10m2 for 2, 3 and 4 and 20m2 for 11 and 12	Copernicus Dataspace – OpenEO.	S2-MBMP

The dependencies contained in the environment.yml file are outlined in table 3.

Table3: dependencies required to use the tools.

Name	Version	Description
Python	3.12.2	Programming language to run the code
Jupyterlab	4.1.4	Computing environment for Python
Folium	0.16.0	Generates interactive maps with Leaflet.js.
Pandas	2.2.1	Data manipulation and analysis.
Geopandas	0.14.3	Geospatial data manipulation and analysis.
Matplotlib	3.8.3	Creates map visualizations
Rasterio	1.3.9	Reads and edits raster datasets.
Numpy	1.26.4	Performs numerical computations on arrays
Requests	2.31.0	Used fetch data from web services or APIs.
Rasterstats	0.19.0	Linear regression for brightness correction
OpenEO	0.28.0	API for Earth observation data processing
Shapely	2.0.3	Used to manage points, lines and polygon data

3.3 Setup

Anaconda Navigator, containing the Python programming language and 'Conda', a package manager for creating shareable environments that tools like this can run on will be installed. This includes Jupyter Lab which will be used to run this tool's code.

3.3.1 Anaconda Navigator

To download Anaconda, navigate to https://docs.anaconda.com/anaconda/install/ and follow the instructions of your operating system.

3.3.2 Creating a Conda Environment

In the Anaconda Navigator side bar, click the 'Environments' tab. You will see the installed packages (fig.67).

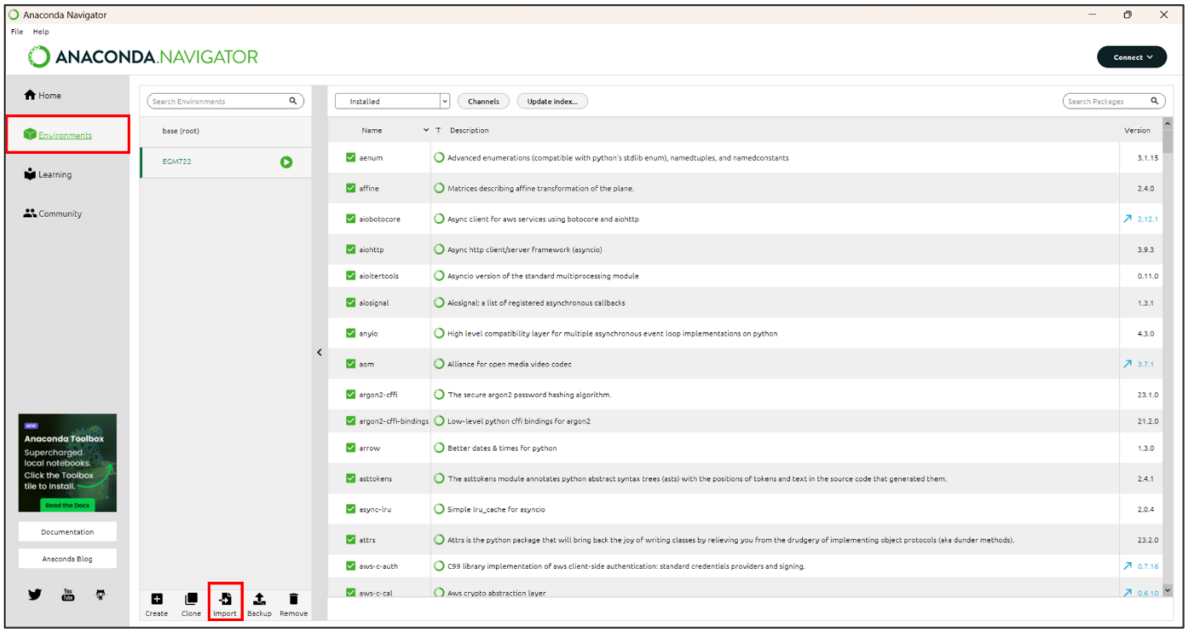

Figure 67: Environments tab of Anaconda Navigator with environments tab and import button highlighted in red.

Next click on the imports tab (fig.3) and select the file 'environment.yml' contained in the .zip file of the tool's download, choosing an appropriate name for the environment (fig. 68).

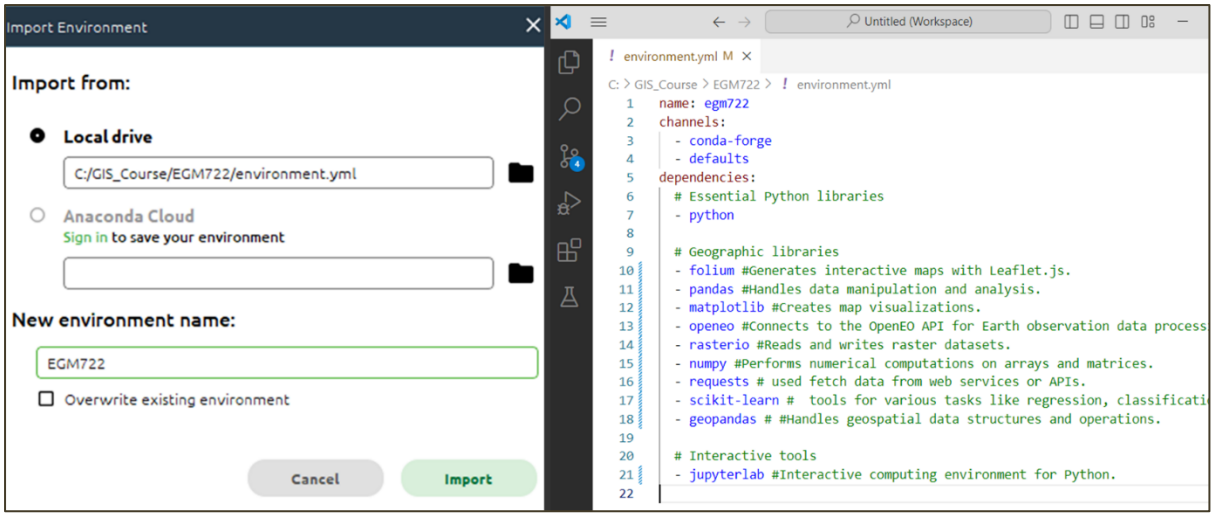

Figure 68: The import config box (left) and the contents of 'environment.yml' (right).

Click Import. The process of installing all the packages may take several minutes. Once finished you will be returned to the environments tab (fig. 67)

Next click on the 'Home' tab in Anaconda Navigator's sidebar (fig. 69).

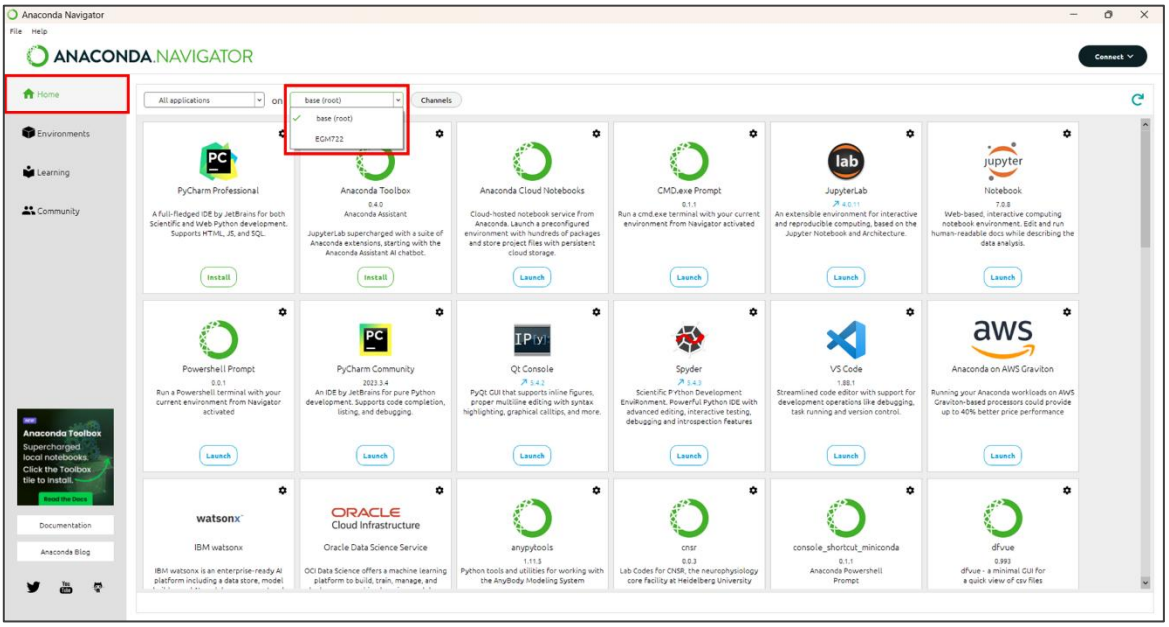

Figure 69: Anaconda Navigator with home tab and environment switching dropdown in red.

The dropdown highlighted in figure 69 should display two options, 'base (root)' and the name of your new environment (in figure 69 this is 'EGM722). **Ensure your environment name is always selected here or the dependencies installed earlier will not be available to it.**

3.3.3 Setting up Jupyter Lab

A configuration file ('.config') needs to be created to change the settings used by Jupyter Lab by default. Launch the CMD.exe prompt (fig.70)

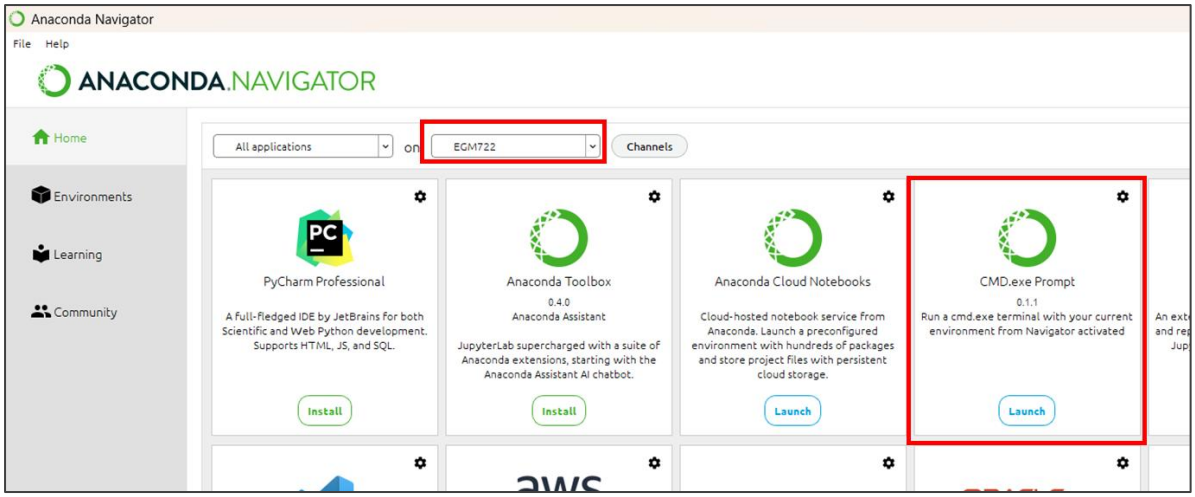

Figure 70: Highlighted locations of selected environment and CMD.exe Prompt

In the command prompt, enter the command:

```
jupyter lab --generate-config
```

This will create a new folder in your user directory called '.jupyter' containing a python script juptyer_lab_config.py. On Windows this is usually 'C:\Users\<your_username>'.

Jupyter lab will by default open in your user directory. Due to security restrictions, it is not possible to navigate to the parent directory of the launch location. So if Jupyter launches in 'C:\Users\RockyBalboa, it is not possible to move to 'C:\Users' or, 'C:\EGM722'. If the directory you are keeping your data in is outside your user directory, you will need to change the default opening folder to your data directory.

This location is also where you should store the following files and folder:

- S2_Plume_Finder
- The folder "Data"
- Environment.yml

If your data directory is in your user directory, you should be able to click and navigate there using the interface of Jupyter Lab. If that is not the case, you will need to do the following:

Open an Anaconda Navigator CMD.exe prompt and type the following command:

```
jupyter --paths
```

This will show something like figure 71 although your file paths will be unique to you.

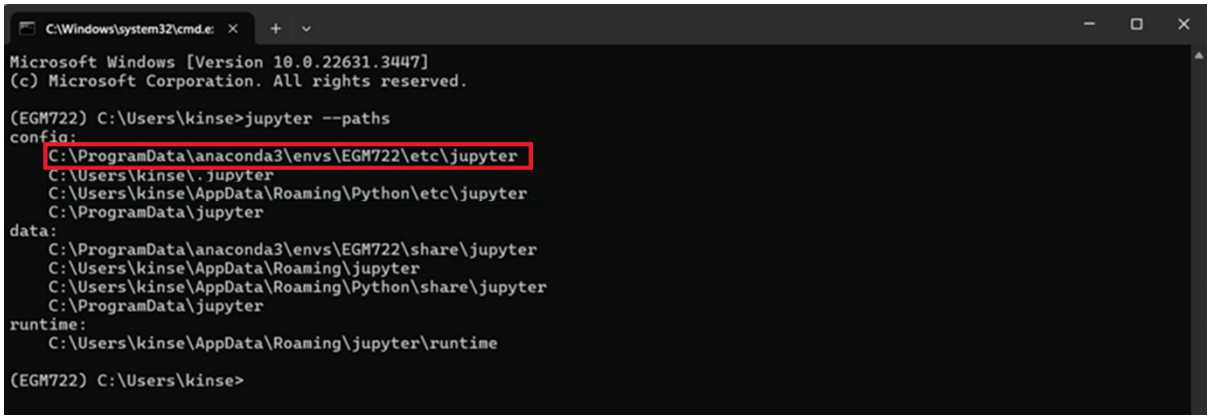

Figure 71: results of 'jupyter –paths' command showing path used by environment highlighted in red.

The 'jupyter_lab_config.py' file mentioned earlier needs to be copy pasted into that folder.

Once 'jupyter_lab_config.py' file has been moved, open it in Notepad++, Visual Studio Code or if you don't have those, Notepad. Using the shortcut 'CTRL + F' type in the following line: 'c.ServerApp.root_dir' (without quotes) and you should find the section highlighted in figure 72.

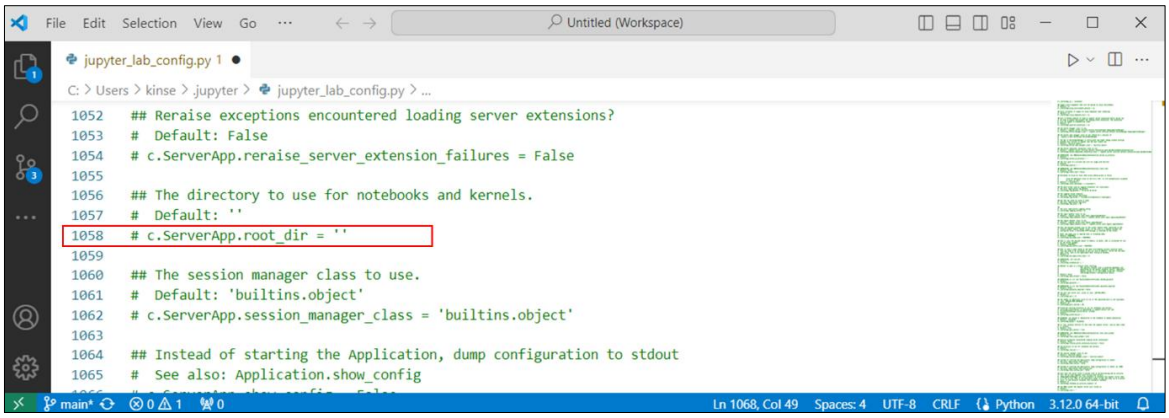

Figure 72: location of 'c.ServerApp.root_dir' in jupyter_lab_config.py

Remove the '#' and space from the start and add the path used by your environment between the quote marks, adding a 'r' beforehand (fig.73).

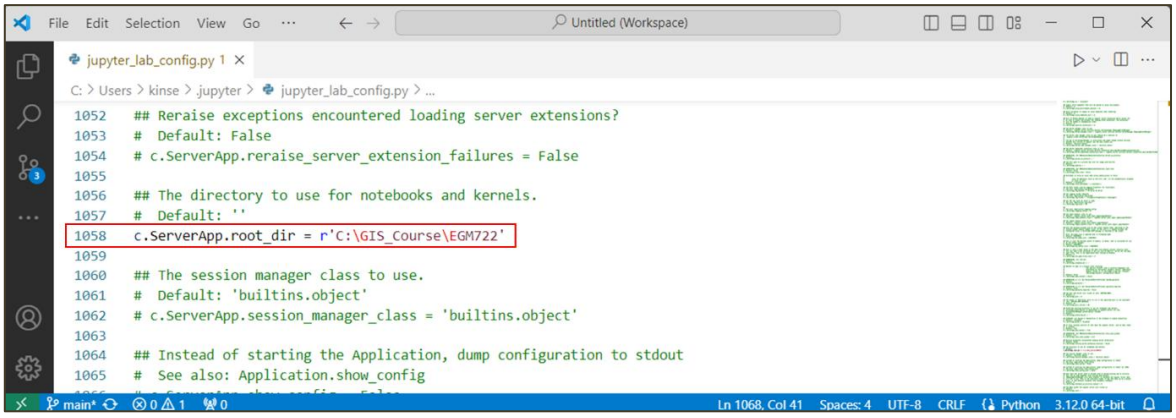

Figure 73: path to data directory added to jupyter_lab_config.py

Save and close this file and return to the Anaconda Navigator 'Home' tab. Launch Jupyter Lab and if you have followed the steps correctly, you should see that your data directory is automatically displayed (fig.74).

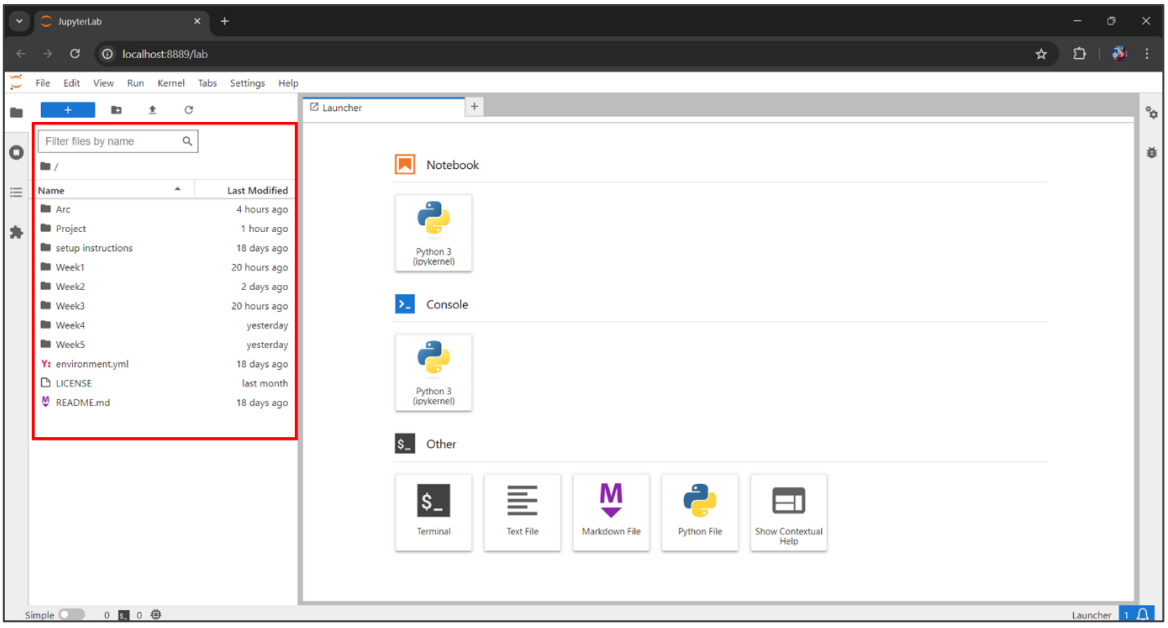

Figure 74: Jupyter Lab showing by default the data directory

3.3.4 OpenEO setup using Anaconda Navigator

OpenEO is an open-source API that allows access to the earth observation satellite missions run by the Copernicus program. These include the satellites used by this tool.

First search in the Anaconda Navigator environments tab for 'openeo'. Make sure that 'Not installed' is selected (fig.75). If the package appears here, click its tick box and select apply. If you can't see it here, please go to section 2.5.

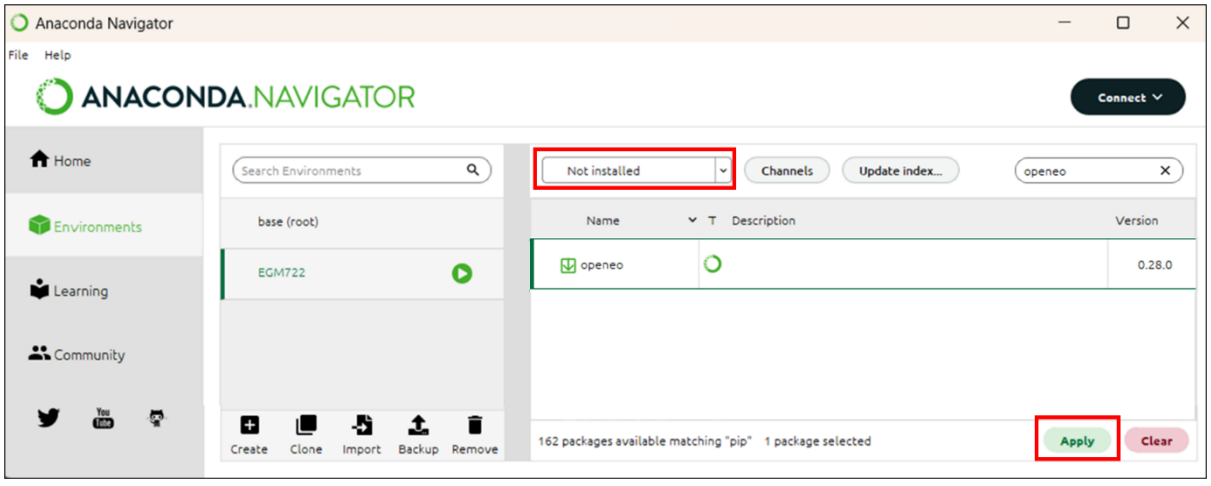

Figure 75: installing OpenEO from Anaconda Navigator.

Next you will be presented with the following screen (fig. 76). One this has finished processing the request. Simply click 'apply' to begin the installation.

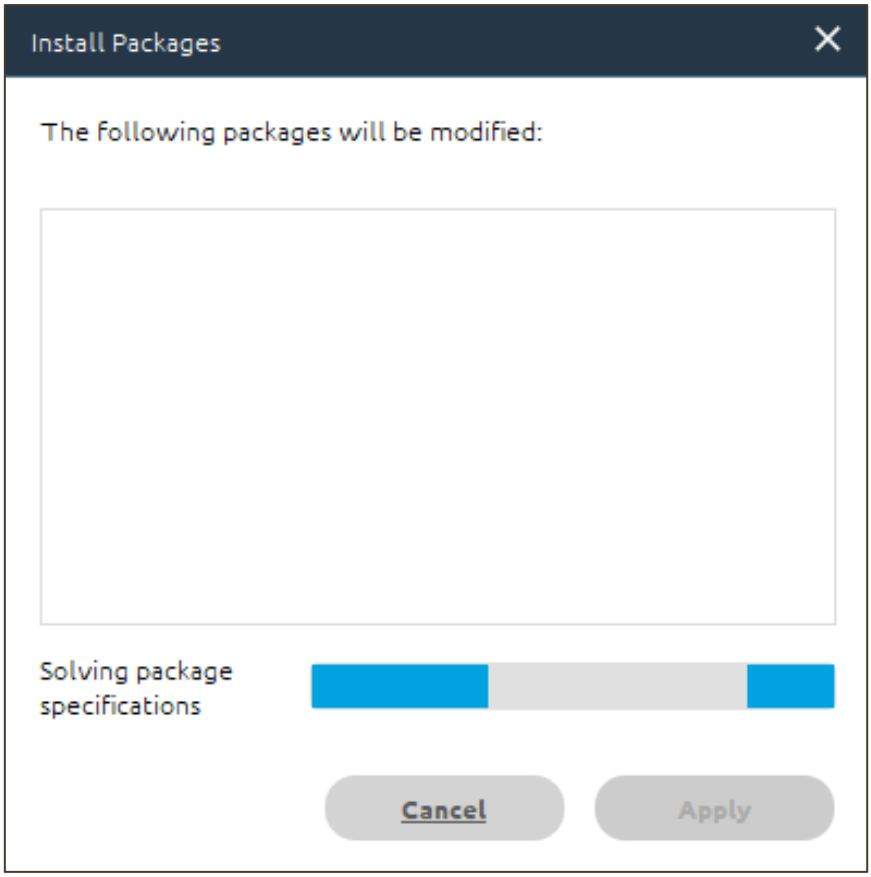

Figure 76: Anaconda Navigator package installer loading screen.

3.3.5 OpenEO setup using PyPi

If Anaconda Navigator cannot find OpenEO you can use PyPi, the official third-party software library for Python. Search for 'pip', selecting the appropriate tick-box and then click apply, then clicking apply once the install packages prompt has finished loading (fig.77).

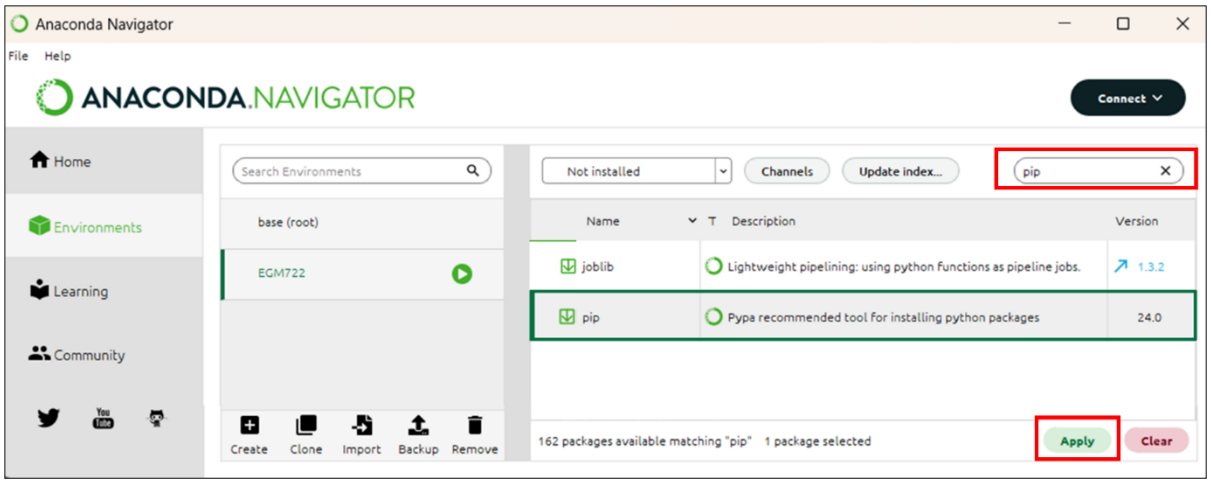

Figure 77: Installing pip via Anaconda Navigator

Open an Anaconda Navigator CMD.exe prompt and type the following command:

```
pip install openeo
```

Once the process has completed, you can close the CMD.exe prompt window.

2.3.6 Registering with Copernicus Data Space Ecosystem.

Accessing OpenEO requires an authentication. To do this, you need to complete a Copernicus Dataspace Registration. Go to https://dataspace.copernicus.eu/ and click the login button (fig. 78)

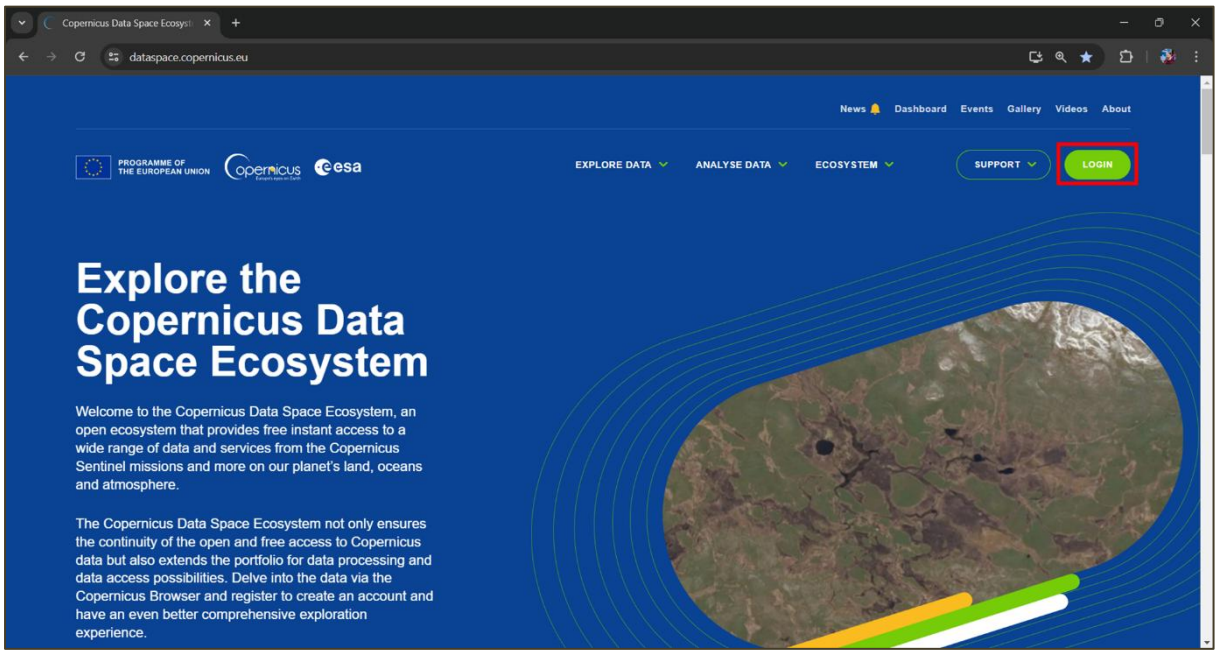

Figure 78: Copernicus Dataspace landing page with login button highlighted in red.

Next click the green 'register' button (fig.79):

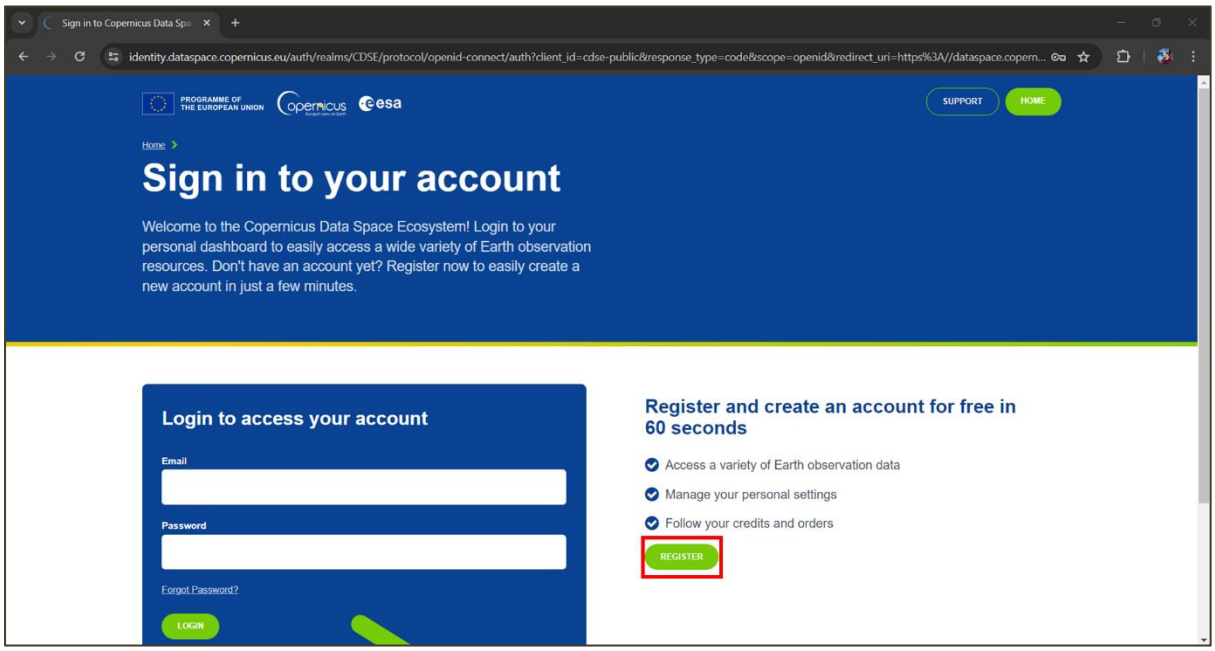

Figure 79: Copernicus Dataspace sign in page.

On the following page, fill out the application form and then at the bottom click the green 'register' button (fig. 80).

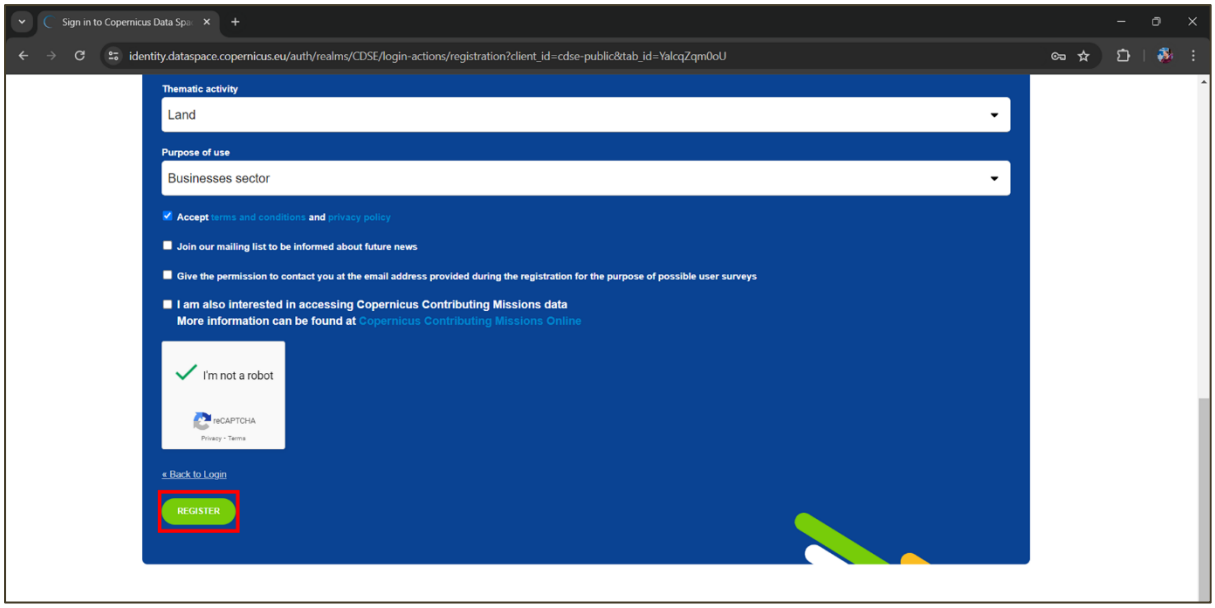

Figure 80: End of Copernicus registration page with register button highlighted in red.

Once registered, you will receive an email asking to verify your address. You can then log-in with your email and chosen password.

For any registration problems, email: help-login@dataspace.copernicus.eu

3.3.7 Running the tools in Jupyter-lab

Now that (almost) everything has been setup you can launch Jupyter Lab in the Anaconda Navigator (fig. 81). Remember as always that your project environment (here 'EGM722') should be selected and not 'base (root)'.

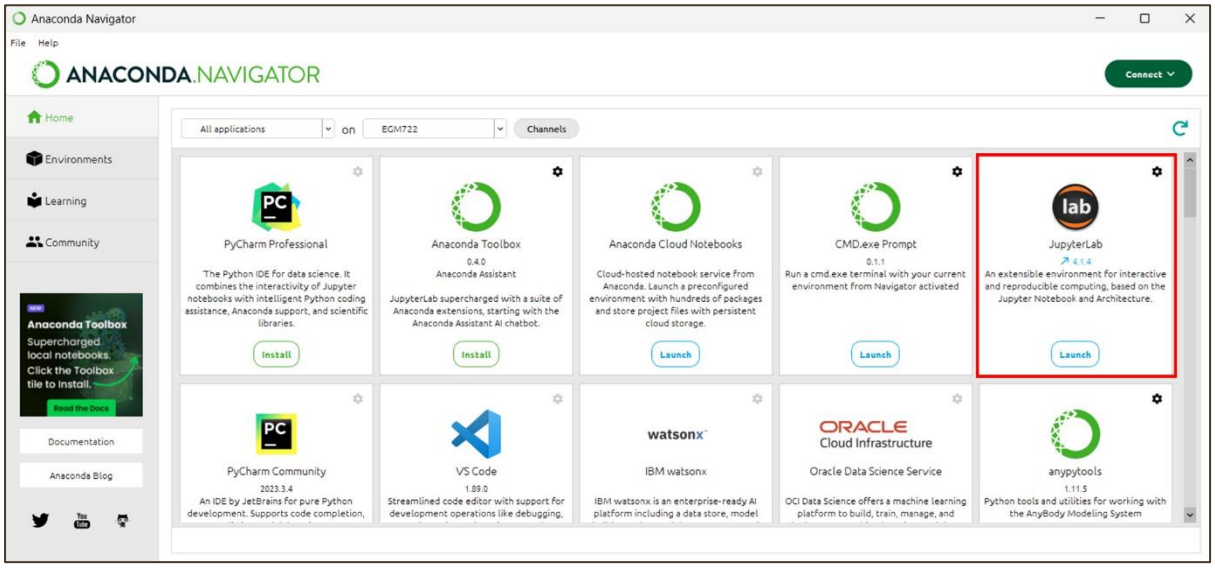

Figure 81: Location of Jupyter Lab in Anaconda Navigator highlighted in red.

Once Jupyter lab opens, you should see the three tools on the left (fig. 82).

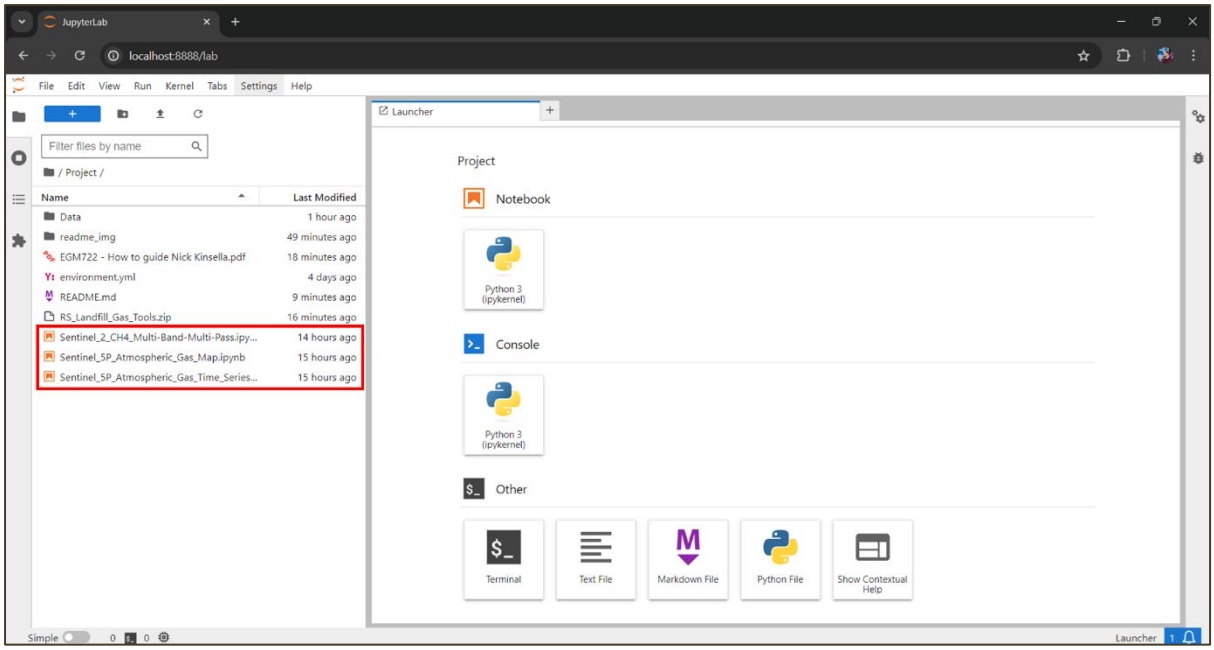

Figure 82: Location of tool scripts in Jupyter lab highlighted in red.

Click on one of the tools to open it. You can then follow the instructions, running the code by clicking the play button (fig. 83).

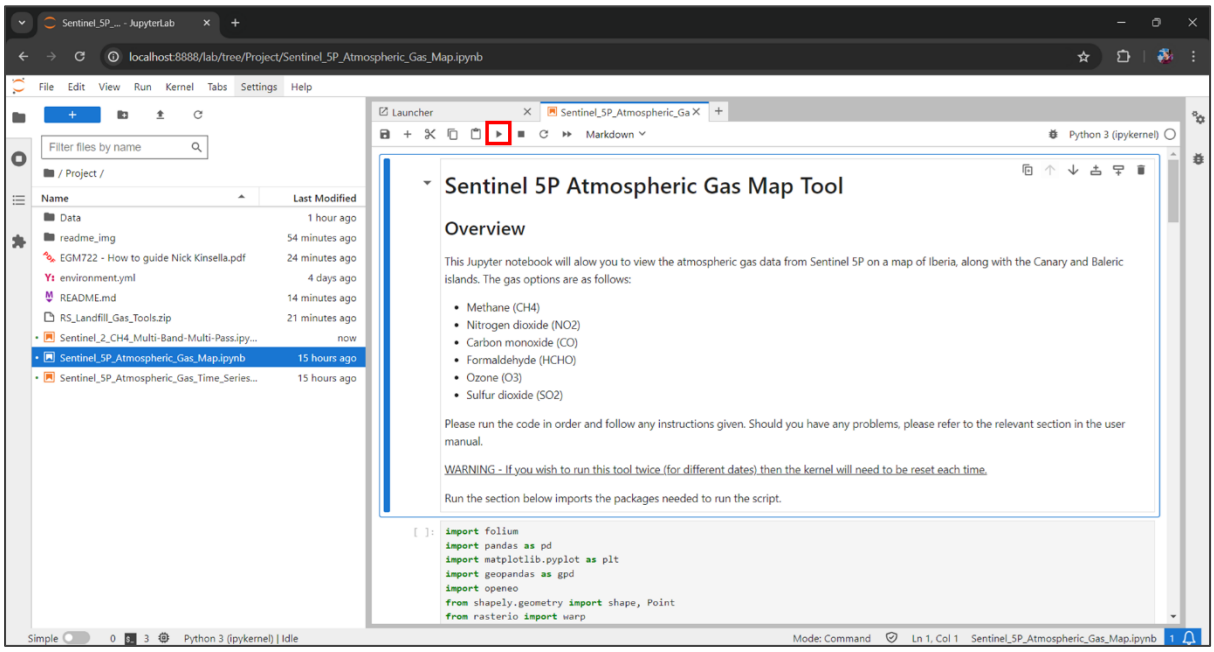

Figure 83: Location of the 'play' button which runs each of the code segments of the workbooks.

3.3.8 Authentication with OpenEO

The very first time one of the tools are run, the following section of code…

```
connection = openeo.connect(url="openeo.dataspace.copernicus.eu")
connection.authenticate_oidc()
```

… will provide you with a URL that will look something like this:

```
Visit https://auth.example.com/device?user_code=EAXD-RQXV to authenticate.
```

Copy this into your web browser and login using the Copernicus Data Space. Once this is complete, run tool's Python script again and it will receive an authentication token, printing the message:

```
Authorized successfully.
```

In future you may be prompted with a new URL to create a new authentication token, whereby you should repeat the steps of this section.

3.3.9 Installing CDS API

The Climate Data Store (CDS) Application Program Interface (API) will provide necessary atmospheric wind information that will enable the software to determine plume flow rate, enabling us to calculate the emission rate, known as 'gas flux'. First we need to register on the European Centre for Medium-Range Weather Forecasts website. In a browser, navigate to https://www.ecmwf.int/ and click 'Log in' (fig. 84).

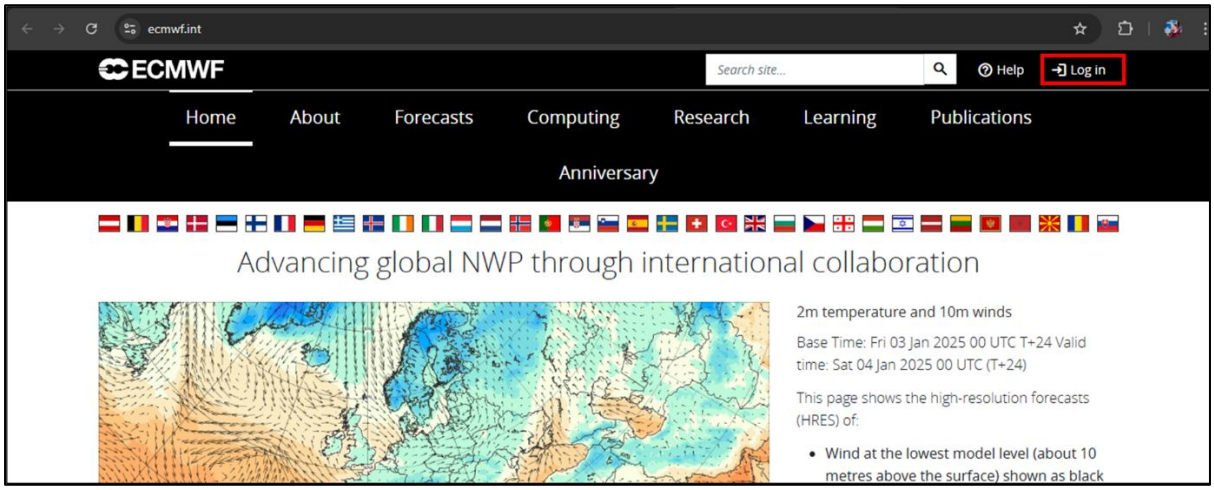

Figure 84: Login button for Copernicus Climate Data Store

Next click register (fig. 85).

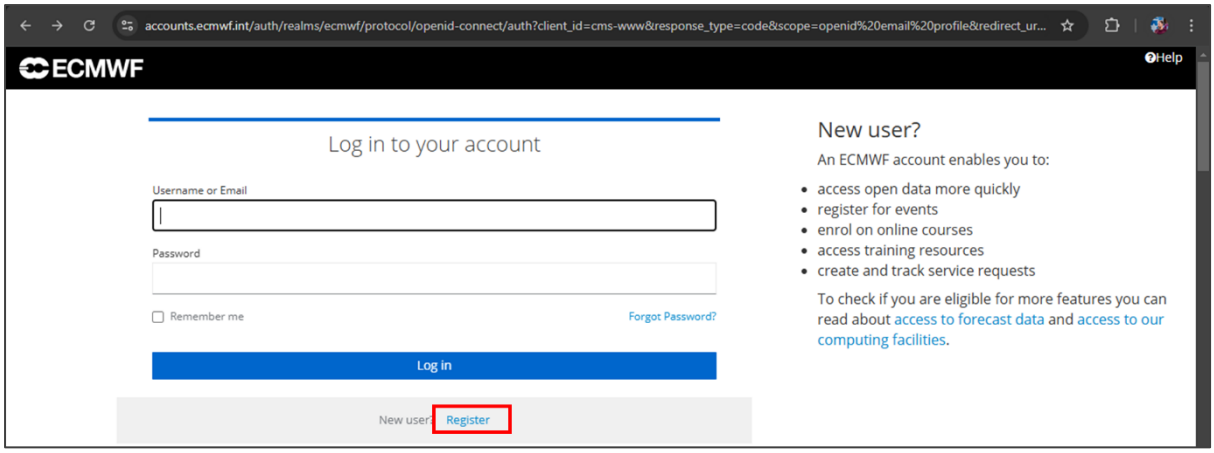

Figure 85: registration button on ECMWF website.

In the next screen, enter your details and click 'register at the bottom of the form.

After you will be prompted to confirm your registration email address. Simply open your email and click the link provided. This needs to be done within 15 minutes or the link will expire (fig. 86).

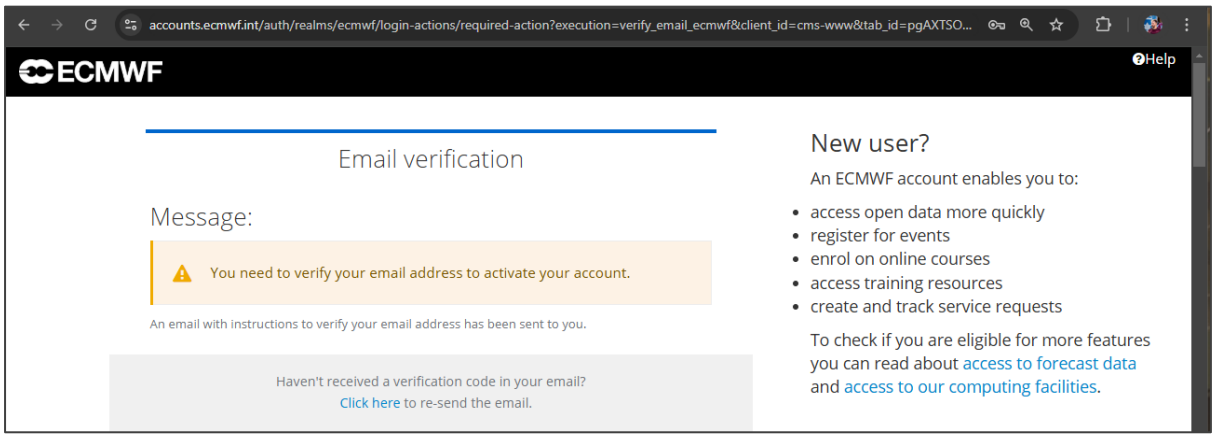

Figure 86: registration button on ECMWF website.

The credentials we created are going to be used on a different website, the Copernicus climate data store. In a web browser, navigate to the following website: https://cds.climate.copernicus.eu/ and click the 'Login – Register' button in the top right of the page (fig. 87)

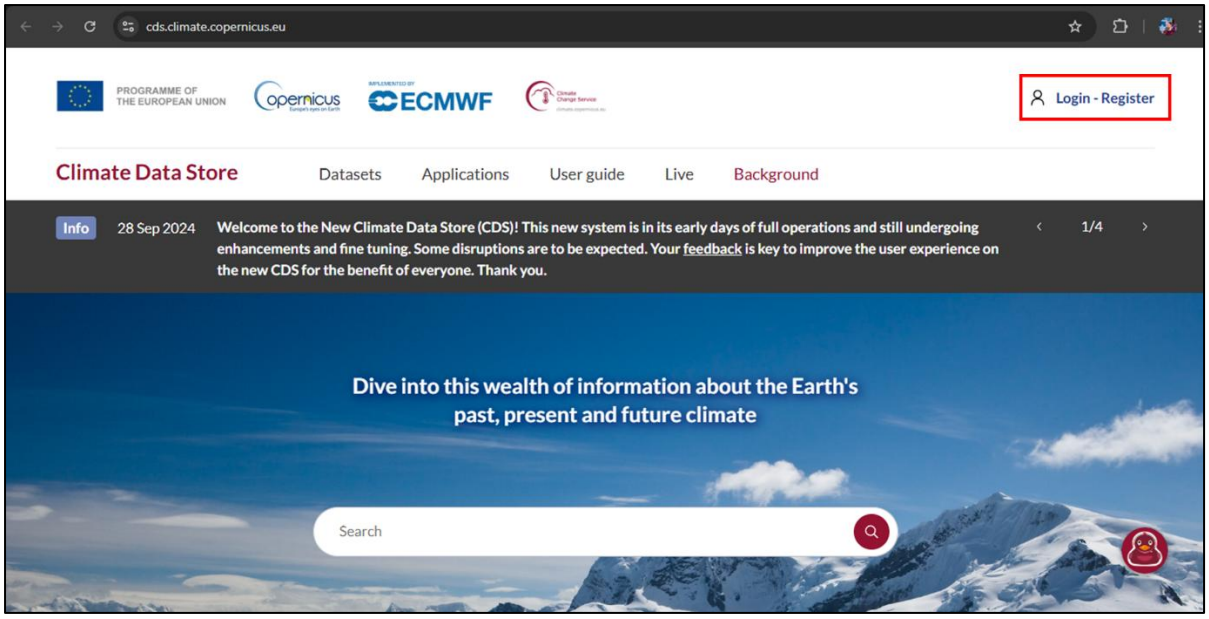

Figure 87: registration button on ECMWF website.

You will see a prompt telling you to set up the credentials on the ECMWF website that we already did a few moments ago. Click 'I understand' (fig. 88).

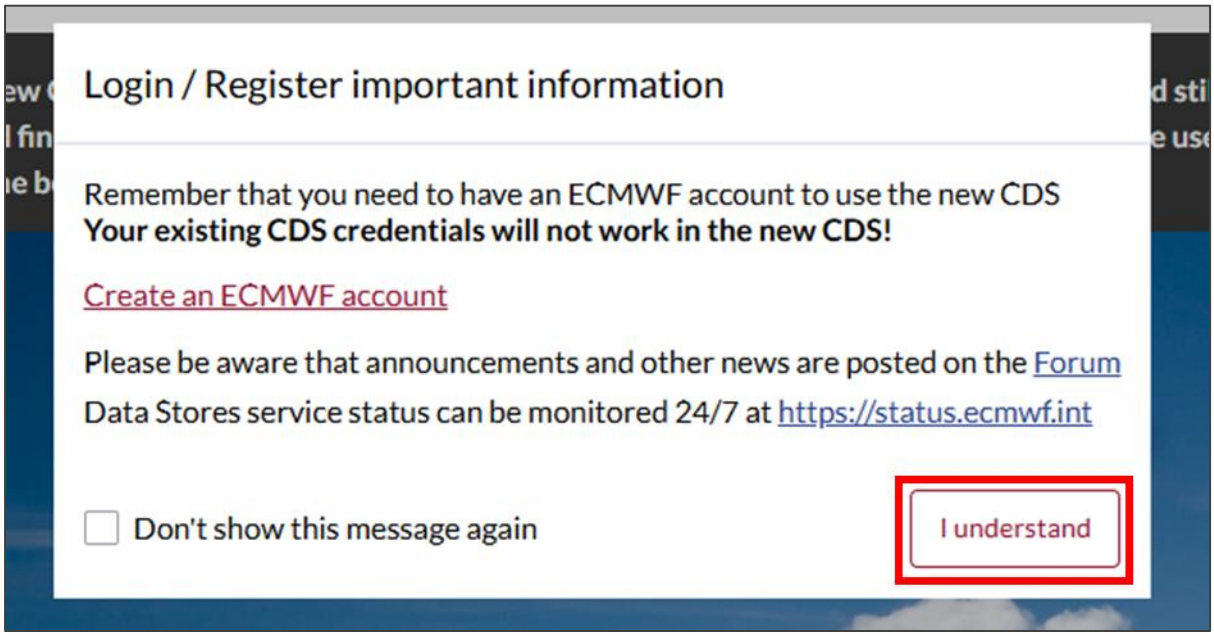

Figure 88: ECMWF registration warning on CDS website.

You should find that you are automatically partly logged into the CDS website (fig. 89).

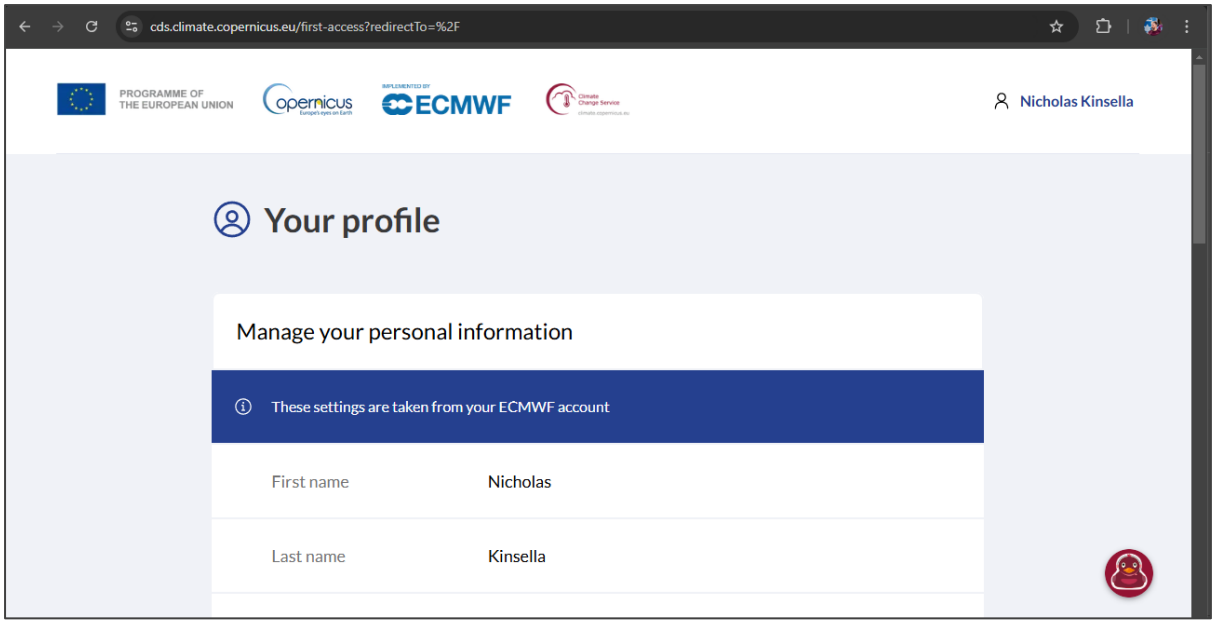

Figure 88: ECMWF registration warning on CDS website.

To fully activate the account you will need to fill in additional information further down, agree to the data protection and privacy statement, the terms of use, and finally and click the 'activate your profile' button at the bottom of the page (fig. 89).

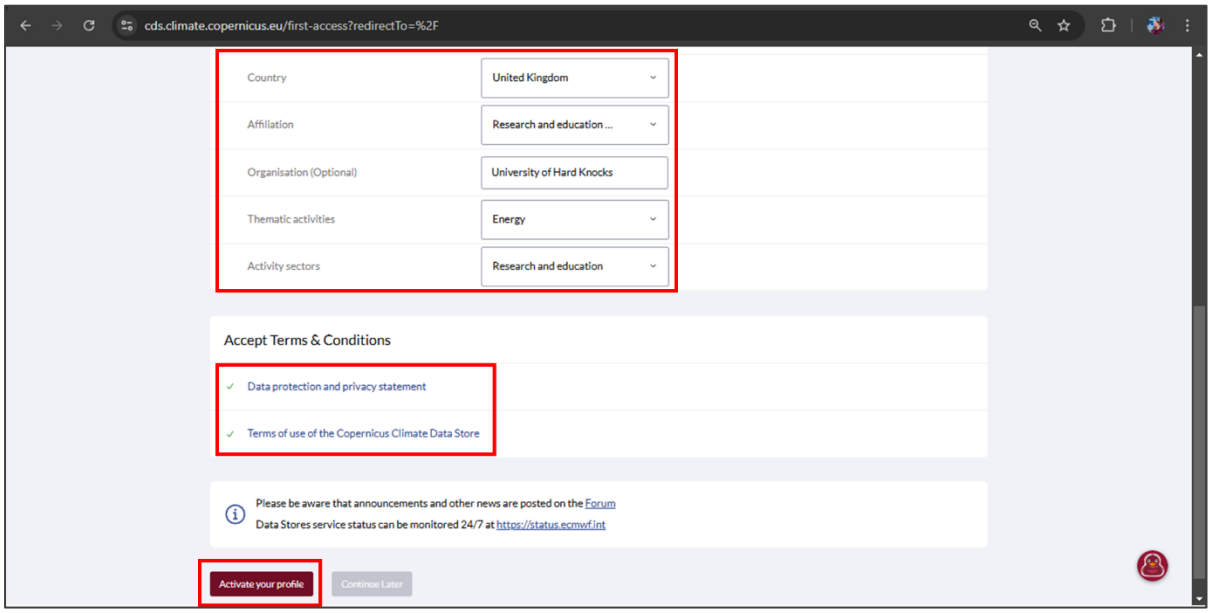

Figure 89: CDS website profile activation.

Open Anaconda Navigator, select your environment from the dropdown menu as always and launch a Command.exe prompt (fig. 90)

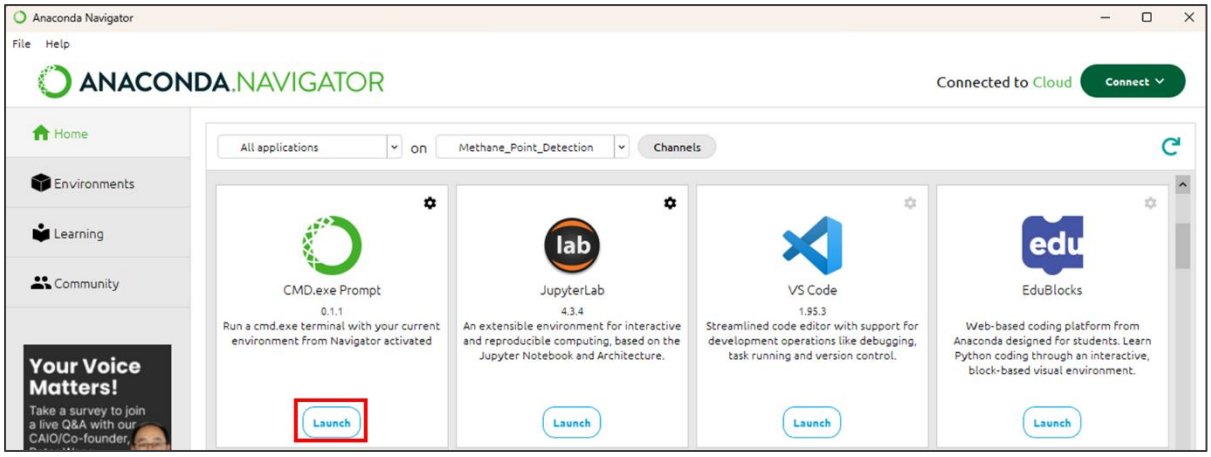

Figure 90: Anaconda Navigator home screen with CMD.exe launch button highlighted.

Enter the command 'conda install cdsapi' without quotes. This will install the CDS API (fig. 91)

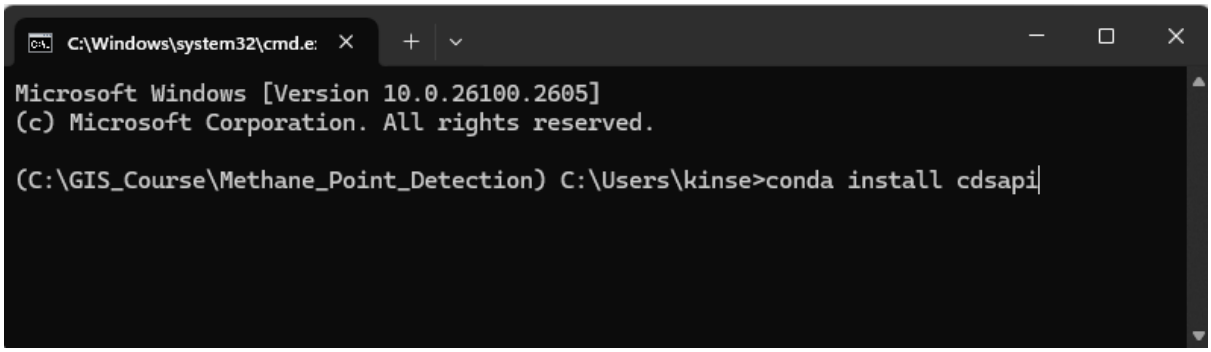

Figure 91: conda install cdsapi command in CMD.exe prompt.

The preparation will take a couple of minutes to complete. Once it is completed, you will be prompted if you would like to proceed with the installation, type 'y' and hit enter. The API will begin installing (fig. 92).

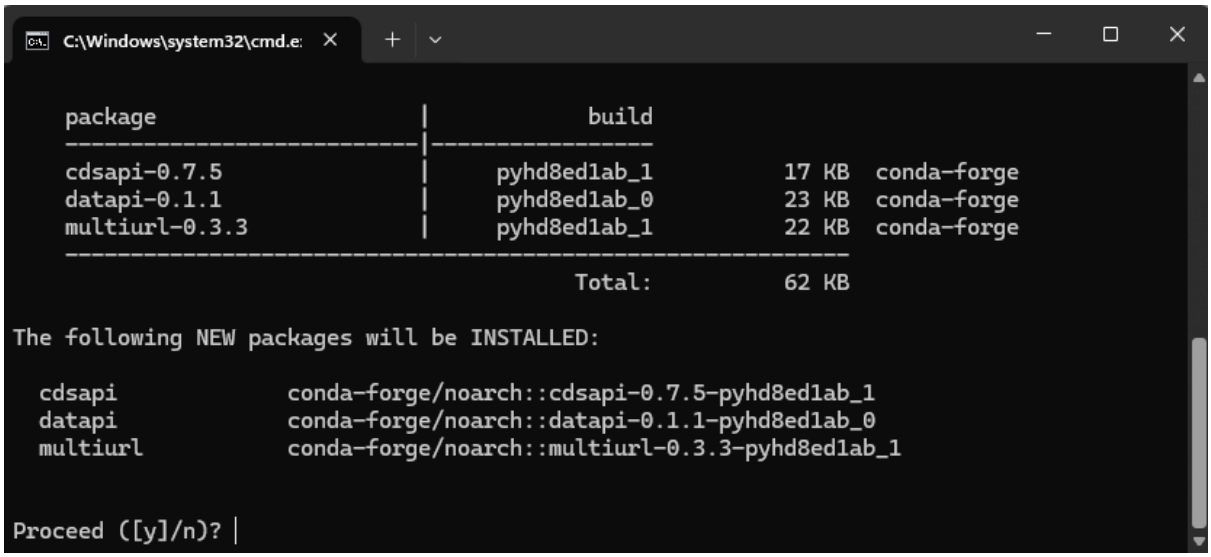

Figure 92: proceed with installation screen in CMD.exe prompt.

Once it is complete, you can close the CMD.exe window.

Before using the CDS API we need to set up the log-in credentials. Navigate the following page https://cds.climate.copernicus.eu/how-to-api and scroll down to the section called 'Setup the CDS API personal access token'. There you will see a URL and an access token in a grey box (fig. 93)

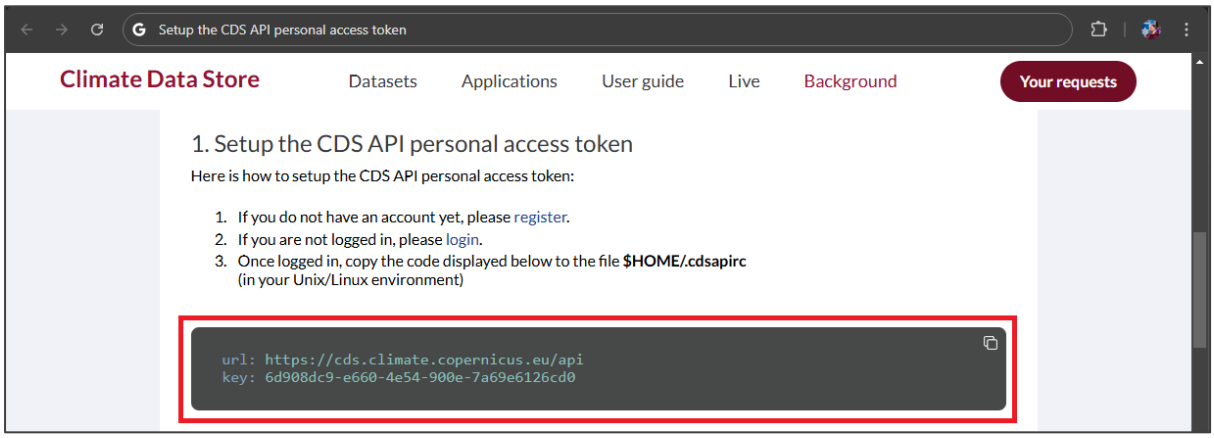

Figure 92: proceed with installation screen in CMD.exe prompt.

As the instructions on the page say, these need to be saved in a file called .cdsapirc in your home directory, for example: 'C:\Users\<YourUsername>\.cdsapirc'. It is important that the file has no extension like .txt. Microsoft notepad unfortunately adds an .txt extension to files by default, even if you don't specify this is what you want to do. Therefore a free program like VS Code should be used instead.

You can download VS Code from https://code.visualstudio.com/. On the landing page, click the button that says, "download for windows" and run the downloaded VSCodeUserSetup.exe file to install the software, accepting all the default options (fig. 93).

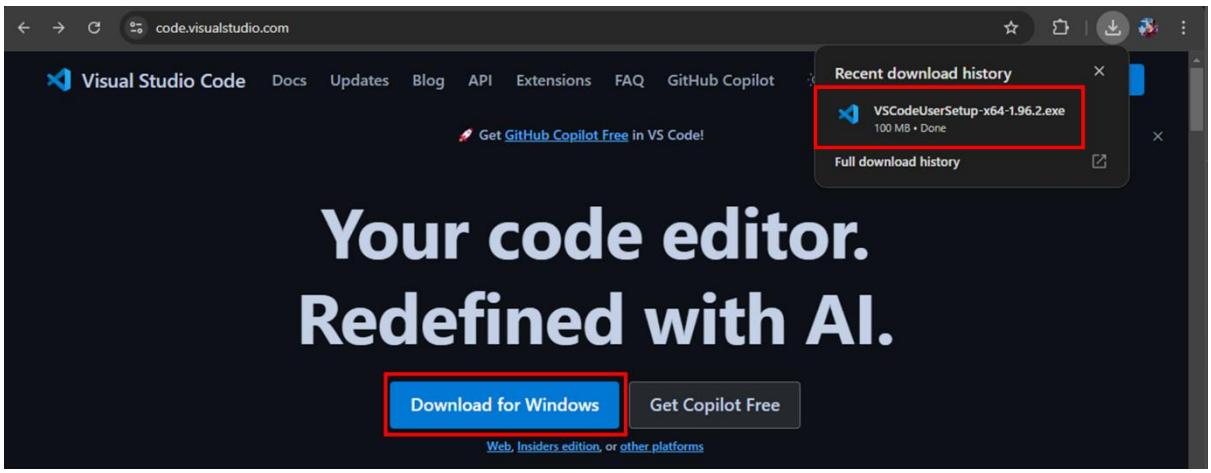

Figure 93: VS Code landing page with download button and downloaded file highlighted.

In VS Code, select file > New Text File (fig. 94)

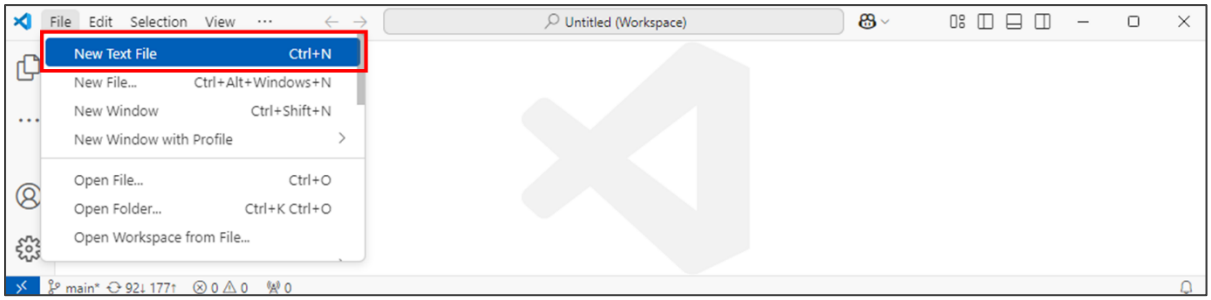

Figure 94: VS Code file > New Text File .

Then paste the credentials from the grey box as shown in figure 92. (fig. 95)

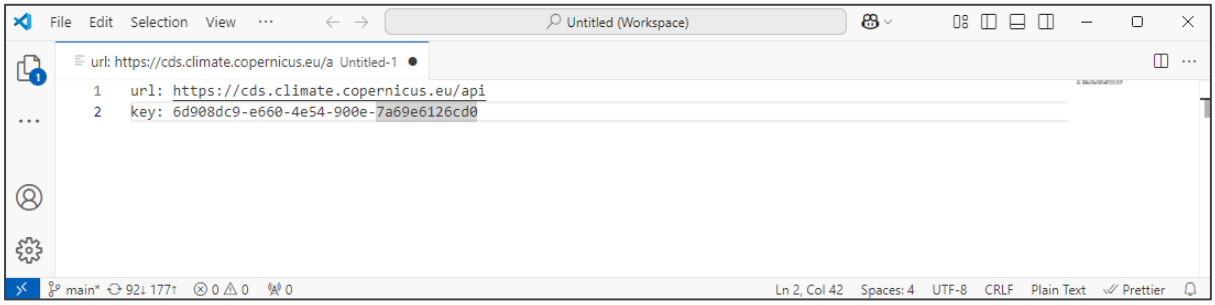

Figure 95: VS Code with CDS API Login credentials

Finally save this to 'C:\Users\<YourUsername>\' as '.cdsapirc' ensuring that 'no extension' is chosen for the file type (fig. 96).

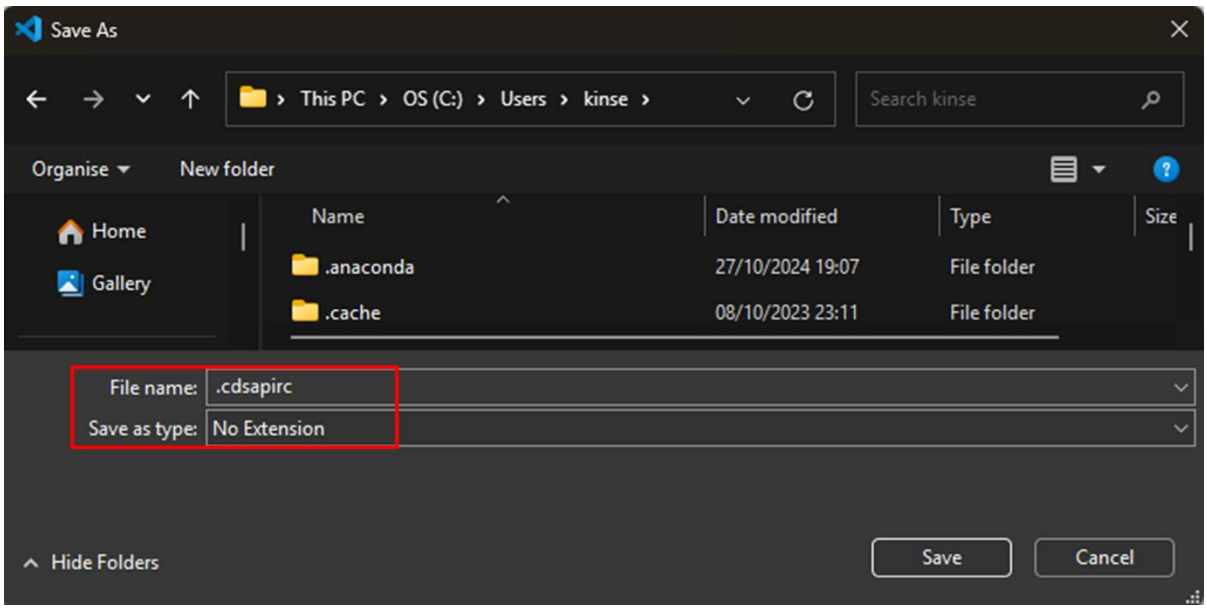

Figure 96: Saving .cdsapirc file with 'No Extension' selected

The first time you run the Sentinel-2 Plume Finder code the following error will occur telling you that the required licences haven't been accepted.

```
HTTPError: 403 Client Error: Forbidden for url: https://cds.climate.copernicus.eu/api/retrieve/v1/processes/reanalysis-era5-single-levels/execution
required licences not accepted
Not all the required licences have been accepted; please visit https://cds.climate.copernicus.eu/datasets/reanalysis-era5-single-levels?tab=download#manage-licences to accept the required licence(s).
```

Figure 97: Initial error code when trying to access wind speed data for field.

The following page will appear. Ensure the 'reanalysis' box is ticked (fig. 98).

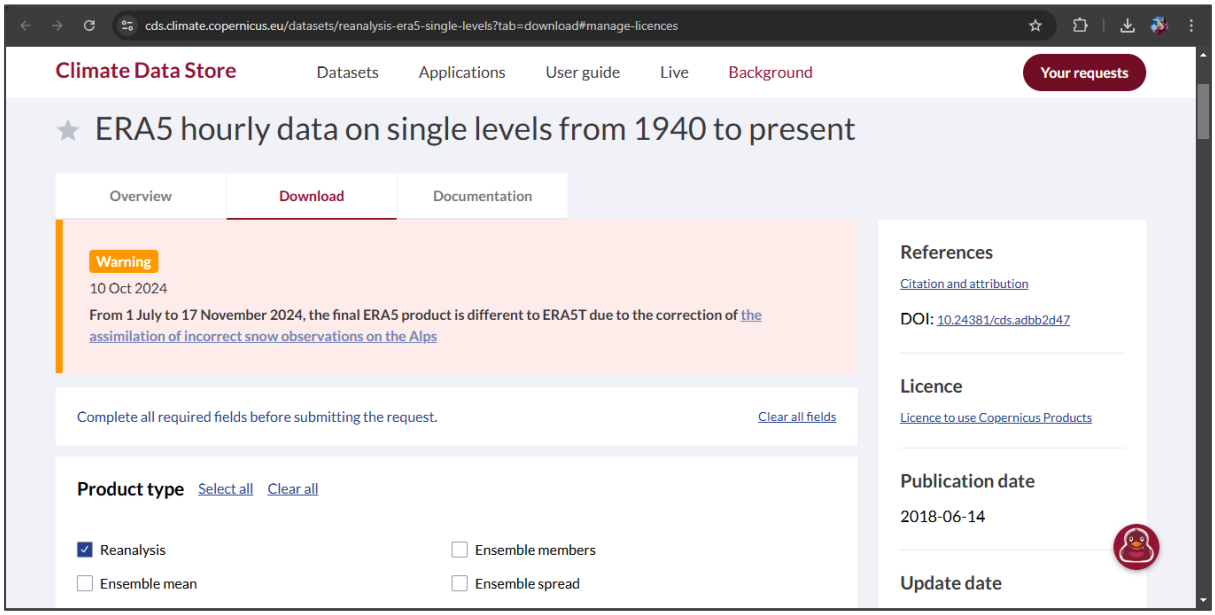

Figure 98: ERA5 licence agreement landing page

Scroll down until you see a menu item titled 'wind', expand the menu and select the tick boxes for '10m v-component of wind', which represents north/south windspeed, and 10m 'u-component of wind', which represents east/west windspeed (fig. 99).

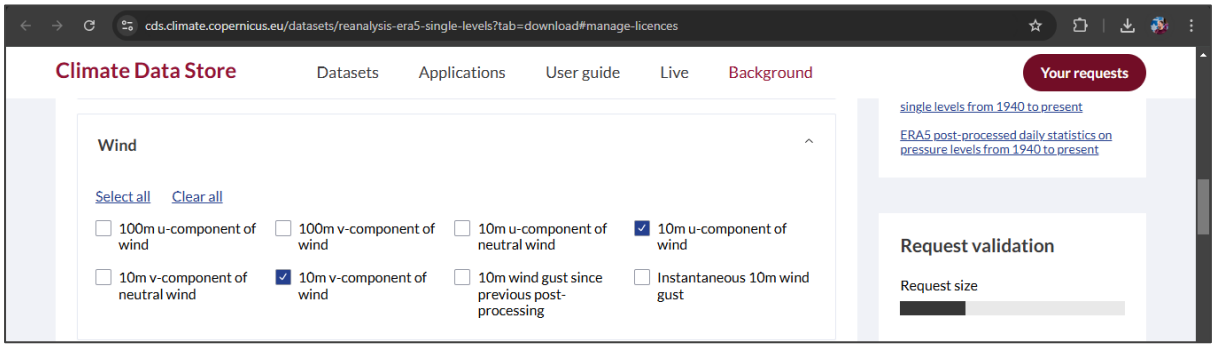

Figure 99: ERA5 licence agreement landing page wind submenu

Further down you will see the Year, Month, Day and Time expandable menus. Here you should select the date periods you are interested in (fig. 100).

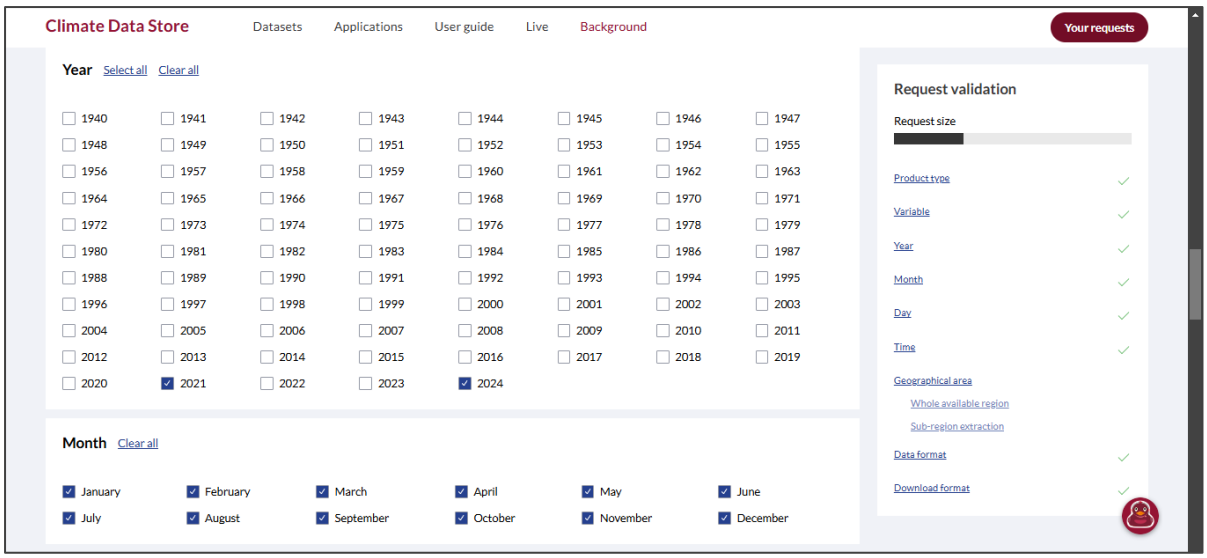

Figure 100: ERA5 licence page with Year and Month submenus. Day submenu is not shown.

Sentinel-2 operates with a sun-synchronous orbit, meaning that it passes over the earth at a similar time of day, every day, to ensure consistent lighting conditions. In Algeria the overpass time is between 10:25am and 10:35am, so we will choose the wind conditions at 10am for our analysis (fig. 101).

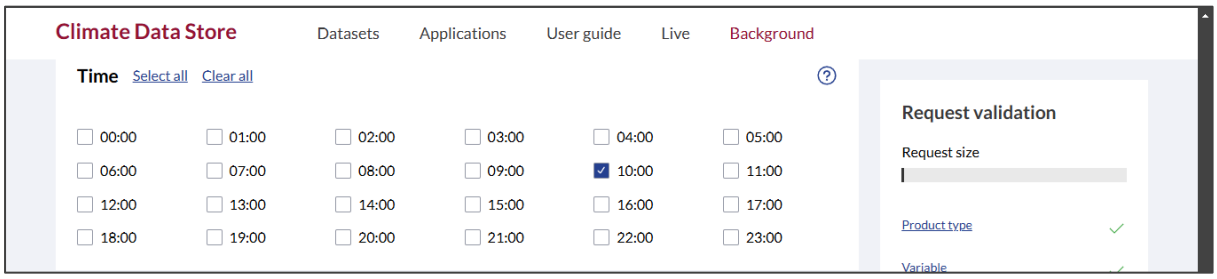

Figure 101: ERA5 licence page with time submenu.

The Geographical area submenu by default has the entire globe selected. For Algeria we want to use the boundings showing in figure 102.

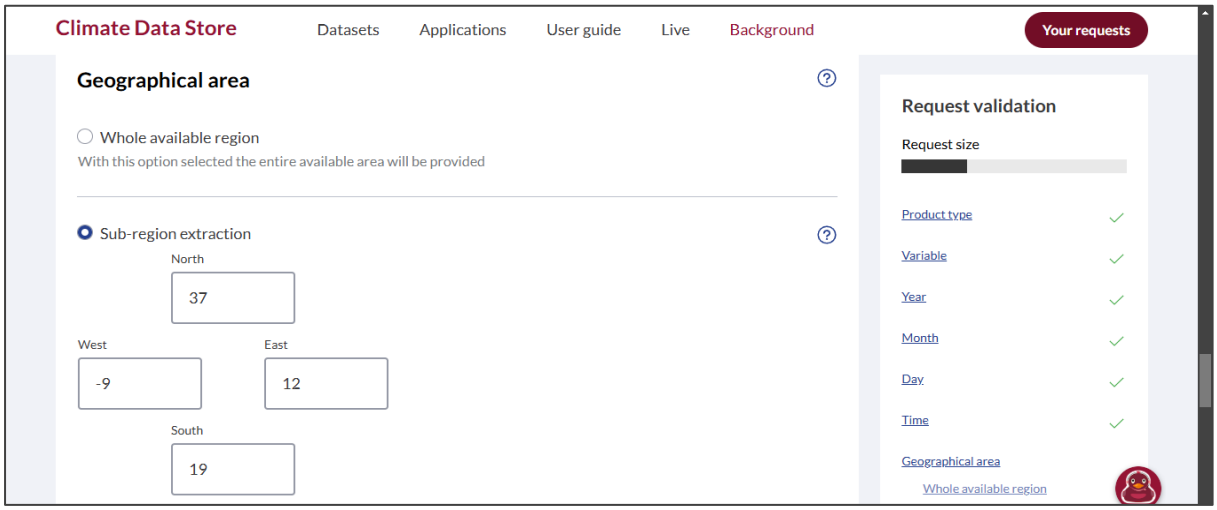

Figure 102: ERA5 licence agreement page, Geographical area submenu.

Choose NetCDF4 for the data format and unarchived for the download format. Click accept terms (fig. 103).

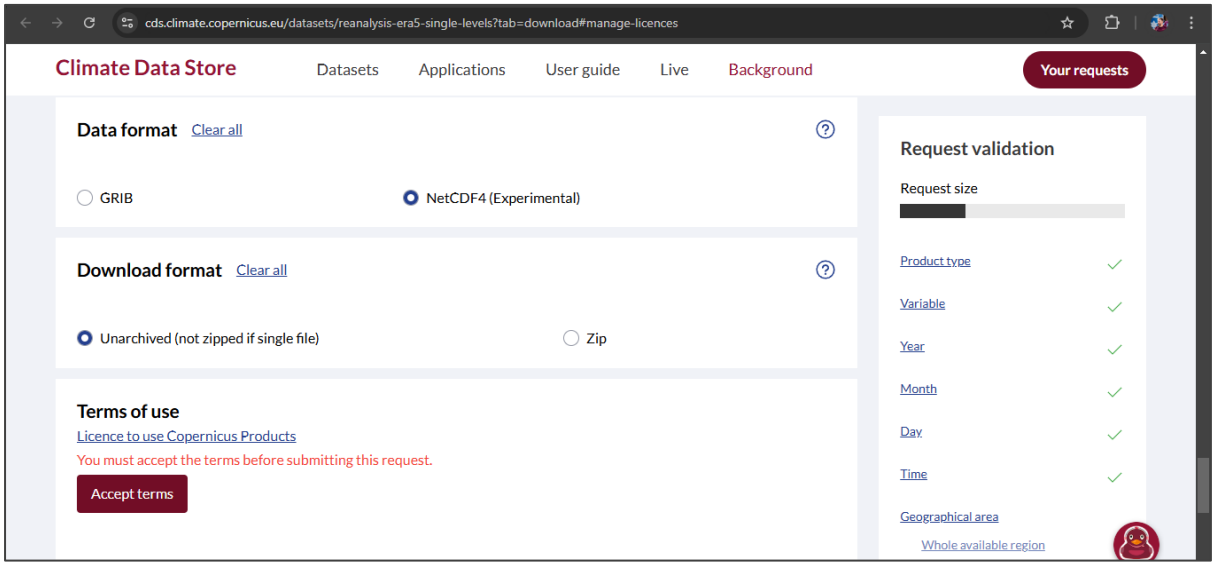

Figure 103: ERA5 licence agreement page, Data format, download format submenus and Accept Terms button.

Finally click 'Submit form' under the API request submenu (fig. 104)

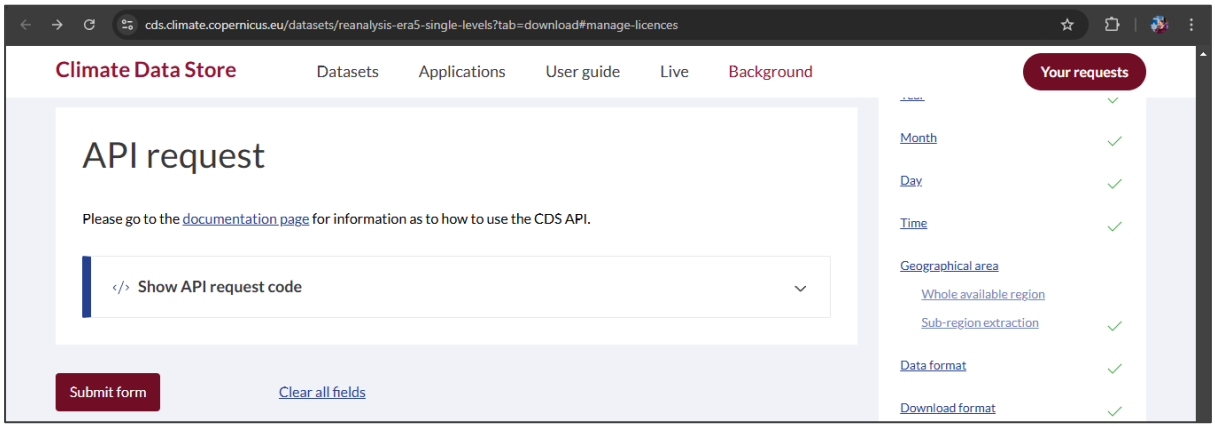

Figure 104: ERA5 licence agreement page, API request submenu and Submit form button.

3.4 Sentinel-2 Plume Finder Code

Overview

This notebook provides a comprehensive workflow for detecting and analysing methane plumes from oil and gas facilities using Sentinel-2 satellite data. It combines satellite imagery processing, wind speed analysis, and regression modelling to estimate methane emission rates accurately. The tool is currently configured for Algeria but could be repurposed for other regions with a few small changes. Key functionalities include:

1. **SWIR Analysis**: Uses Sentinel-2's Short-Wave Infrared (SWIR) bands to detect methane plumes.
2. **Plume Tagging**: Uses user driven tagging of plume locations.
3. **Regression-Based Emission Estimation**: Employs a XGBoost regression model to estimate methane emission rates based on plume characteristics and wind speed data.
4. **Dynamic Model Updates**: Facilitates the addition of new training data to refine the XGBoost model for improved predictions.
5. **Plume measurement**: Creates an interactive map to visualise SWIR-derived plumes, and provides an estimate of their emission rates in a table in kg/h.

This tool is designed for researchers, policymakers, and environmental analysts aiming to quantify and monitor methane emissions efficiently. Code box 1 loads the dependencies for the tool to run.

```
# Connecting to Sentinel-2 data
import openeo

# Available Date finder imports
import requests
import pandas as pd

# SWIR and Truecolour processing imports
import numpy as np
import geopandas as gpd
import rasterio
from rasterio.features import geometry_mask
from rasterio.warp import calculate_default_transform, reproject, Resampling

# Interactive Maps and Visualisation
import folium  # For creating interactive maps
from folium import Map, LayerControl, LatLngPopup, Rectangle  # Map features
from folium.raster_layers import ImageOverlay  # Overlay raster images on maps
from folium import FeatureGroup  # For grouping map layers
import matplotlib.pyplot as plt  # For plotting and visualisation

# Wind Speed imports
import cdsapi
from tempfile import NamedTemporaryFile
import xarray as xr

# Plume analysis imports
from scipy.ndimage import label  # For segmentation and labelling of regions
```

```
from scipy.spatial import ConvexHull  # For calculating convex hulls of shapes
from scipy.spatial.distance import pdist

# Imports related to predictive model
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from xgboost import XGBRegressor
```

Code 1: Loading of dependencies for the code to run.

Connect to OpenEO

Code 2 below establishes a connection with the Copernicus openEO platform which provides a wide variety of earth observation datasets

- If this does not read as 'Authorised successfully' or 'Authenticated using refresh token', then please ensure that you have completed the setup steps as outlined in section 2.3.6 of the how to guide.
- If you have followed the steps in section 2.3.6 correctly and the problem persists, please look at https://dataspace.copernicus.eu/news for any information about service interruptions.
- If there is no news of service problems you can raise a ticket here: https://helpcenter.dataspace.copernicus.eu/hc/en-gb/requests/new

```
connection = openeo.connect(url="openeo.dataspace.copernicus.eu")
connection.authenticate_oidc()
```

Code 2: Connecting to OpenEO

Display field names.

Code 3 loads the oil and gas field list. Hassi Messaoud is site 86. If you are interested in a different field, please look-up its id number.

```
studysite_csv = pd.read_csv(r'C:\GIS_Course\Methane_Point_Detection\Sentinel-
2_Algeria_Methane\Data\Algerian_Oil_and_Gas_Fields.csv')
pd.set_option('display.max_rows', None)
print(studysite_csv.to_string(index=False))
```

Code 3: Displaying the contents of the site file for easy reference.

Site Selection

Code box 4 below, specify the field number we are interested in for analysis.

site_id = 86

```
site_id = 86  # Specify the oil and gas field ID for the field you want to examine.
# Retrieve the name of the field from the dataset
field_name = studysite_csv[studysite_csv['id'] == site_id].iloc[0]['name']
# Confirmation message
print(f"Site {site_id} ({field_name}) loaded correctly.")
```

Code 4: Displaying the contents of the site file for easy reference.

Multi-Band Multi-Pass Analysis

The method for CH_4 plume visualisation was taken from Varon et al. (2021a) who outlined three connected approaches for detecting CH_4 columns using Sentinel-2 data: The first, Single-Band Multi-Pass (SBMP), utilised the Short-Wave Infrared band 12 (SWIR-2) on a day with a known emission, and then compared it to a day when there was no emission. Second was a Multi-Band Single-Pass method (MBSP) which took the Short-Wave Infrared band 11 (SWIR-1) and SWIR-2 readings for the same day, and extracted the CH_4 plume by least squares fitting, and then differencing the band datasets. The final method was a Multi-Band Multi-Pass method (MBMP), which effectively combines the two previous approaches by differencing SWIR-1 and SWIR-2 for a day with a known emission and then again for a day with the emission absent. The MBMP method produced the best results with the least emission artefacts and so was chosen for this study (Varon et al., 2021a).

Scenes with a maximum of 5% cloud cover were chosen for both the non-emission scene and active-emission scenes. No emission and active emission scenes were processed using the multi-band-single-pass equation as outlined in Varon et al. (2021a):

$$MBSP = \frac{B12 - B11}{B11}$$

Where:

- B12 is the Sentinel-2 SWIR-2 band.
- B11 is the Sentinel-2 SWIR-1 band.

Once active emission and no emission scenes have been calculated, the following equation is used to calculate the multi-band-multi-pass raster.

$$MBMP = ActiveMBSP - NoMBSP$$

Where:

- ActiveMBSP is the multiband single pass for the active emission scene
- NoMBSP is the multiband single pass for the no emission scene.

The active emission scene and no emission scene are considered in this analysis to be one satellite pass apart unless there is a large amount of interference from features such as clouds or other plumes, in which case an earlier date should be selected.

A final step in this analysis that has been added to scale the MBMP dataset mean to zero and all other valid pixel values by that amount. This has been done to account for seasonal variations in solar radiation levels that may affect the measurements of this tool.

To begin this process we need to determine what days have available satellite data.

Available dates for the analysis.

Sentinel 2 provides data approximately once every 2 - 3 days, so not every date you can input is valid. The code below will tell you what dates are available to use for the oil/gas field of your choice.

The one parameter you need to modify before running the code is:

- **temporal_extent** = ["2020-01-01", "2020-01-31"] (change this to your chosen date range using "YYYY-MM-DD" format.)

Once you have done this run code box 5 and the available dates should appear below in a matter of seconds. This section of code was provided by Sonneveld, E. (2024).

```
# Specify the date range you want to check for available data.
temporal_extent = ["2020-01-01", "2020-01-31"]

def get_spatial_extent(site_id):
    site = studysite_csv[studysite_csv['id'] == site_id].iloc[0]
    return {
        "west": site['west'],
        "south": site['south'],
        "east": site['east'],
        "north": site['north']
    }

def fetch_available_dates(site_id, temporal_extent):
    spatial_extent = get_spatial_extent(site_id)
    catalog_url =
f"https://catalogue.dataspace.copernicus.eu/resto/api/collections/Sentinel2/search.json?
box={spatial_extent['west']}%2C{spatial_extent['south']}%2C{spatial_extent['east']}%2C{s
patial_extent['north']}&sortParam=startDate&sortOrder=ascending&page=1&maxRecords=1000&s
tatus=ONLINE&dataset=ESA-
DATASET&productType=L2A&startDate={temporal_extent[0]}T00%3A00%3A00Z&completionDate={tem
poral_extent[1]}T00%3A00%3A00Z&cloudCover=%5B0%2C{cloud_cover}%5D"
    response = requests.get(catalog_url)
    response.raise_for_status()
    catalog = response.json()
    dates = [date.split('T')[0] for date in map(lambda x: x['properties']['startDate'],
catalog['features'])]
    return dates
cloud_cover = 5

available_dates = fetch_available_dates(site_id, temporal_extent)
print("Available dates:", available_dates)
```

Code 5: Code for determining available dates for analysis

Choosing the "Active Emission" Date

Code 6 chooses a so-called active emission date must be chosen from one of the available datasets. This will be the chosen day we are looking for plumes.

Like before, the one parameter you need to modify before running the code is:

temporal_extent = ["2020-01-17", "2020-01-17"]

Change this to your chosen date range using "YYYY-MM-DD" format.

Please note that the temporal extent dates MUST BE IDENTICAL because we are only choosing a single date.

If you receive an error message of 'NoDataAvailable' then please check the list of available data above and try again.

```
active_temporal_extent = ["2020-07-30", "2020-07-30"] # Enter parameters for the active
emission day

def active_emission(site_id, active_temporal_extent):
    site = studysite_csv[studysite_csv['id'] == site_id].iloc[0]

    active_emission = connection.load_collection(
        "SENTINEL2_L2A",
        temporal_extent=active_temporal_extent,
        spatial_extent={
            "west": site['west'],
            "south": site['south'],
            "east": site['east'],
            "north": site['north']
        },
        bands=["B11", "B12"],
    )
    active_emission.download("Sentinel-2_active_emissionMBMP.Tiff")

active_emission(site_id, active_temporal_extent)
```

Code 6: Code for downloading active emission dataset.

Choosing the "No Emission" Date

Code 7 chooses the no emission date using the same process. This is the dataset we will compare the "Active Emission" one too. The recommended choice is the satellite overpass immediately before the "Active Emission" one.

So if your active emission day is 2020-01-17, your no emission day would be 2020-01-14

In an ideal world, the "No Emission" day should contain no emissions, but in fields with a lot of activity like Hassi Messaoud, this may not be possible. Such an instance will not cause problems in most cases. The emissions for these dates will simply appear as dark clouds on the SWIR data and can be ignored.

The one parameter you need to modify before running the code is:

temporal_extent = ["2020-01-14", "2020-01-14"]

The temporal extent dates MUST BE IDENTICAL

If you receive an error message of 'NoDataAvailable' then please check the list of available data above and try again.

```
no_temporal_extent = ["2020-07-22", "2020-07-22"] # Enter parameters for the active
emission day

def no_emission(site_id, temporal_extent):
    site = studysite_csv[studysite_csv['id'] == site_id].iloc[0]

    no_emission = connection.load_collection(
        "SENTINEL2_L2A",
        temporal_extent=no_temporal_extent,
        spatial_extent={
            "west": site['west'],
            "south": site['south'],
            "east": site['east'],
            "north": site['north']
        },
        bands=["B11", "B12"],
    )
    no_emission.download("Sentinel-2_no_emissionMBMP.Tiff")

no_emission(site_id, no_temporal_extent)
```

Code 7: Code for downloading active emission dataset.

Downloading a Background Satellite Image

Code 8 helps with locating the source of the emission by displaying a true colour satellite image of the oil/gas field that the data will be superimposed over. This will help distinguish between true emissions and visual spectrum observable clouds. It is recommended that you choose the same date as your active emission.

```
"""The true-colour raster needs to be reprojected to WGS84 line up correctly with the
folium map.
The same will be done later for the SWIR dataset.
"""
def reproject_to_epsg4326(data, meta):
    target_crs = "EPSG:4326"

    # Calculate transform and metadata for the target CRS
    transform, width, height = calculate_default_transform(
        meta['crs'], target_crs, meta['width'], meta['height'], *meta['bounds']
    )

    # Update metadata for the new projection
    new_meta = meta.copy()
    new_meta.update({
        "crs": target_crs,
        "transform": transform,
        "width": width,
        "height": height,
    })

    # Prepare an array for reprojected data
    reprojected_data = []
    for i in range(meta['count']):
        # Create an empty numpy array to store the reprojected data for the band
        destination = np.empty((height, width), dtype=data[i].dtype)
        reproject(
            source=data[i],
            destination=destination,
            src_transform=meta['transform'],
            src_crs=meta['crs'],
            dst_transform=transform,
            dst_crs=target_crs,
            resampling=Resampling.nearest
        )
        reprojected_data.append(destination)

    return np.array(reprojected_data), new_meta  # Returning as numpy array instead of
writing to file


# The truecolour download uses the same date as the active_emission function.
def truecolour_image(site_id, temporal_extent):

    site = studysite_csv[studysite_csv['id'] == site_id].iloc[0]

    truecolour_image = connection.load_collection(
        "SENTINEL2_L2A",
        temporal_extent=temporal_extent,
        spatial_extent={
```

```
            "west": site['west'],
            "south": site['south'],
            "east": site['east'],
            "north": site['north']
        },
        bands=["B02", "B03", "B04"],
    )

    # Download the true colour image
    file_path = "Sentinel-2_truecolourMBMP.Tiff"
    truecolour_image.download(file_path)

    # Read the file into memory
    with rasterio.open(file_path) as src:
        data = [src.read(i) for i in range(1, src.count + 1)]
        meta = src.meta.copy()
        meta['bounds'] = src.bounds

    # Reproject the data in memory
    reprojected_data, reprojected_meta = reproject_to_epsg4326(data, meta)

    # Return reprojected data and metadata
    return reprojected_data, reprojected_meta

# Run and store the reprojected image as a variable
temporal_extent = active_temporal_extent
reprojected_image_data, reprojected_image_meta = truecolour_image(site_id,
temporal_extent)
```

Code 8: Code for downloading true colour satellite image for data visualisation

Running Plume Visualiser Analysis

Code 9 will use the satellite data to display plumes above 2000kg/h in ideal conditions. Provided all the variables above have been run correctly, this next section should take moments to complete.

```
# to align the datasets on the folium map
def get_bounds(site_id, csv_path):
    df = pd.read_csv(csv_path)
    site = df[df['id'] == site_id]
    if site.empty:
        raise ValueError(f"Site ID {site_id} not found in the CSV file.")
    site = site.iloc[0]
    return [[site['south'], site['west']], [site['north'], site['east']]]

csv_path = r'C:\GIS_Course\Methane_Point_Detection\Sentinel-
2_Algeria_Methane\Data\Algerian_Oil_and_Gas_Fields.csv'
bounds = get_bounds(site_id, csv_path)

# File path definitions
Active_Multiband = "Sentinel-2_active_emissionMBMP.Tiff"
No_Multiband = "Sentinel-2_no_emissionMBMP.Tiff"
output_file = "SWIR_diff.tiff"
masked_output_file = "SWIR_diff_masked_urban.tiff"
urban_geojson = r"C:\GIS_Course\Methane_Point_Detection\Sentinel-
2_Algeria_Methane\hassi_messaoud_urban.geojson"

# The main MBMP calculations begin here
with rasterio.open(Active_Multiband) as Active_img, rasterio.open(No_Multiband) as
No_img:

    # These divisions convert the Sentinel-2 L1C digital numbers to reflectance data.
    Active_B11 = Active_img.read(1).astype(float) / 10000.0
    Active_B12 = Active_img.read(2).astype(float) / 10000.0
    No_B11 = No_img.read(1).astype(float) / 10000.0
    No_B12 = No_img.read(2).astype(float) / 10000.0

    #This perfoms two MBSP calculations, one for each satelite pass.
    MBSP_active = (Active_B12 - Active_B11) / Active_B11
    MBSP_no = (No_B12 - No_B11) / No_B11

    #This perfoms the MBMP calculation.
    SWIR_diff = MBSP_active - MBSP_no

# Reproject and save SWIR_diff to EPSG:4326 for the folium map
with rasterio.open(Active_Multiband) as src:
    target_crs = "EPSG:4326"
    transform, width, height = calculate_default_transform(
        src.crs, target_crs, src.width, src.height, *src.bounds
    )
    meta = src.meta.copy()
    meta.update({
        "crs": target_crs,
        "transform": transform,
        "width": width,
        "height": height,
        "count": 1,
        "dtype": SWIR_diff.dtype
```

```
    })
    with rasterio.open(output_file, "w", **meta) as dest:
        reproject(
            source=SWIR_diff,
            destination=rasterio.band(dest, 1),
            src_transform=src.transform,
            src_crs=src.crs,
            dst_transform=transform,
            dst_crs=target_crs,
            resampling=Resampling.nearest
        )
"""
Urban areas proved to be a problem whne segmenting the plume from the scene.
These have been masked but this is an imperfect solution as plume length is
a strong predictor of emission rate and this can be cut off by the mask.
It should be assumed that plumes that cross urban areas are underestimated.
"""
# Load GeoJSON and create urban mask
urban_areas = gpd.read_file(urban_geojson)
with rasterio.open(output_file) as src:
    urban_areas = urban_areas.to_crs(src.crs)

    # Rasterize the urban areas
    urban_mask = geometry_mask(
        [feature["geometry"] for feature in
urban_areas.to_crs(src.crs).__geo_interface__["features"]],
        out_shape=(src.height, src.width),
        transform=src.transform,
        invert=True
    )

    # Apply the urban area mask to SWIR_diff
    swir_diff = src.read(1)
    swir_diff_masked = np.where((urban_mask) | (swir_diff == -0.0) | (swir_diff_masked >
3000), -32768, -swir_diff)

    """
    to account for seasonality, the median value of the dataset (i.e. the background)
has been
    adjusted to zero
    """

    target_median = 0.0  # Define target median value

    # Compute the current median (ignoring NoData values)
    current_median = np.median(swir_diff_masked[swir_diff_masked != 32768])

    # Compute shift needed
    shift_value = target_median - current_median

    # Apply shift to all valid pixels
    swir_diff_masked = np.where(swir_diff_masked != 32768, swir_diff_masked +
shift_value, 32768)

    print(f"Adjusting median from {current_median} to {target_median}, shifting by
{shift_value}")
```

```
    # Save the masked SWIR_diff to a new file
    meta = src.meta.copy()
    meta.update(dtype=rasterio.float32, nodata=np.nan)
    with rasterio.open(masked_output_file, "w", **meta) as dest:
        dest.write(swir_diff_masked.astype(rasterio.float32), 1)

# Calculate centre for map using masked SWIR_diff raster bounds
with rasterio.open(masked_output_file) as src:
    map_bounds = src.bounds
    centre_lat = (map_bounds.top + map_bounds.bottom) / 2
    centre_lon = (map_bounds.left + map_bounds.right) / 2

# Create Folium map
m = Map(location=[centre_lat, centre_lon], zoom_start=10, control_scale=True)

# Use the reprojected image stored in memory instead of loading from a file
blue, green, red = reprojected_image_data[0], reprojected_image_data[1],
reprojected_image_data[2]

# this is to adjust the brightness of the truecolour image as it can be a little dark
sometimes.
brightness_factor = 0.03 # only change this number
blue = np.clip(blue * brightness_factor, 0, 255)
green = np.clip(green * brightness_factor, 0, 255)
red = np.clip(red * brightness_factor, 0, 255)

# Stack bands to create RGB image
rgb = np.dstack((red, green, blue))
rgb = rgb / rgb.max()
rgb = np.log1p(rgb)
rgb = rgb / rgb.max()

# Add true colour image overlay
with rasterio.open(masked_output_file) as src:
    swir_bounds = [[src.bounds.bottom, src.bounds.left], [src.bounds.top,
src.bounds.right]]

truecolour_overlay = ImageOverlay(
    name="Truecolour",
    image=rgb,
    bounds=swir_bounds,
    opacity=1,  # Lower opacity for blending with SWIR overlay
    interactive=True,
    zindex=1,  # Lower zindex to place below SWIR overlay
)
truecolour_overlay.add_to(m)

# Load and stretch SWIR_diff for visualization
with rasterio.open(masked_output_file) as src:
    swir_bounds = [[src.bounds.bottom, src.bounds.left], [src.bounds.top,
src.bounds.right]]
    swir_data = src.read(1)

    # Mask invalid data and clip negative values
    swir_data = np.ma.masked_invalid(swir_data)
    swir_data = np.ma.masked_equal(swir_data, 32768)
```

```
    # Calculate mean and std only for valid data
    filtered_swir_data = swir_data[swir_data >= -3000]  # Ignore values below 3000
    mean = np.nanmean(filtered_swir_data)
    std = np.nanstd(filtered_swir_data)
    std_factor = 2  # Stretch factor

    # Calculate stretching bounds within the valid data range
    lower_bound = max(mean - std_factor * std, swir_data.min())
    upper_bound = min(mean + std_factor * std, swir_data.max())

    # Normalize the data to [0, 1]
    normalized_swir_data = np.clip((swir_data - lower_bound) / (upper_bound -
lower_bound), 0, 1)

    # Apply colourmap
    cmap = plt.get_cmap('viridis')
    rgb_data = (cmap(normalized_swir_data.filled(0))[:, :, :3] * 255).astype(np.uint8)

# Add SWIR_diff overlay to map
swir_overlay = ImageOverlay(
    name="SWIR Data",
    image=rgb_data,
    bounds=swir_bounds,
    opacity=1,  # Adjust opacity for visibility
    interactive=True,
    zindex=2  # Ensure SWIR overlay is above other layers
)
swir_overlay.add_to(m)

"""
The next section adds known plume locations as red boxes around the sites.
"""
# Add GeoJSON data as a layer group
vector_point_path = r"C:\GIS_Course\Methane_Point_Detection\Sentinel-
2_Algeria_Methane\Data\known_point_sources.geojson"
gdf = gpd.read_file(vector_point_path)
geojson_layer = FeatureGroup(name="Known Point Sources", show=True)
for _, row in gdf.iterrows():
    lat, lon = row.geometry.y, row.geometry.x
    box_size = 0.002  # Approximate size for 20x20 pixels (adjust if needed)
    bounds = [[lat - box_size, lon - box_size], [lat + box_size, lon + box_size]]
    rect = Rectangle(
        bounds=bounds,
        color="red",
        fill=False,
    )
    geojson_layer.add_child(rect)
geojson_layer.add_to(m)
""" This creates a clickable lat long popup event on the
map that will be used for tagging the plumes"""
LayerControl().add_to(m)
m.add_child(LatLngPopup())
# Display map
display(m)
```

Code 9: Code displaying the map

Plume tagging

Once code 9 has run, a map like the following will appear.

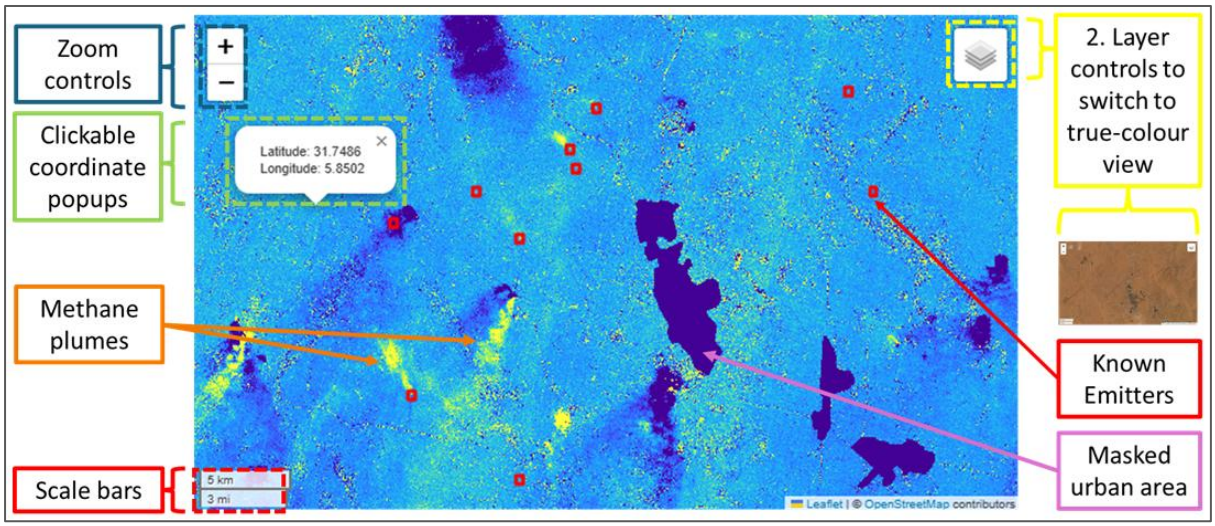

Figure 105: annotated map of code 10's plume visualiser analysis.

To deal with this problem, 11 known plume locations have been programmed into the system which will automatically detect plumes at those locations (Marked as red squares on the map above). Should a plume be located away from these predefined areas, a manual tagging system can be used using the guide below.

Should a plume be located away from these predefined areas, a manual tagging system can be used using figure 106 and table 4 as a guide and code box 10.

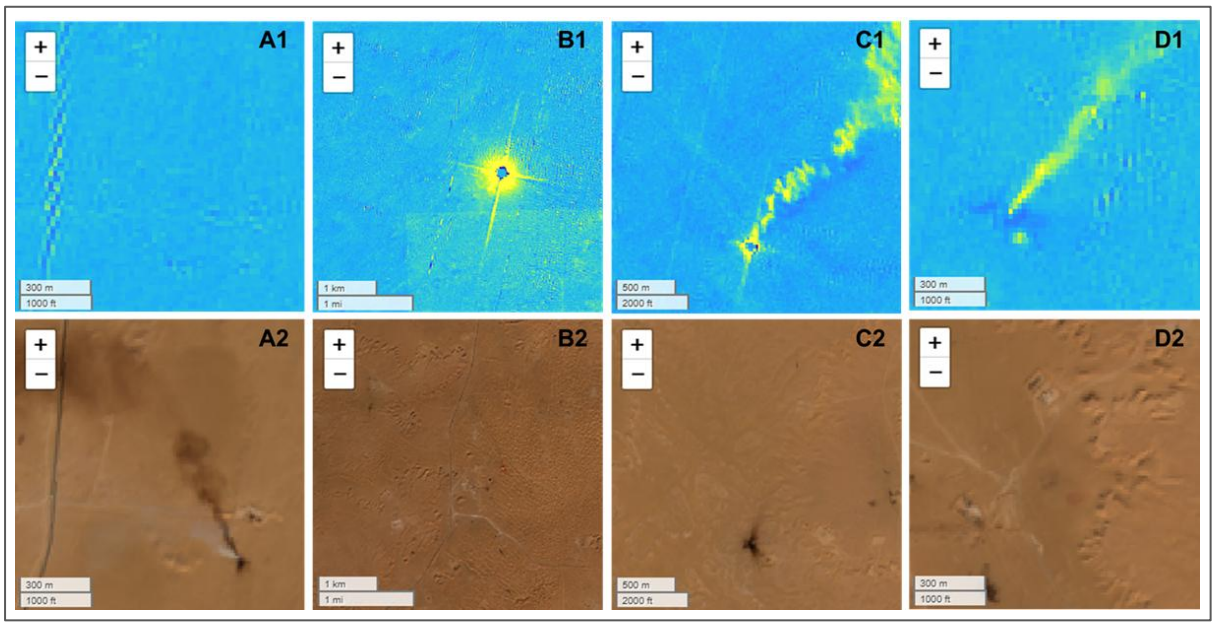

Figure 106: Emission examples. The top row shows the MBMP data and the bottom row shows the same location in true-colour. A = Non-CH_4 plume, B = Flaring with complete combustion, no CH_4 plume, C = Flaring with incomplete combustion with CH_4 plume and D = CH_4 plume with no flaring.

Each of these scenarios are explained further by the matrix in table 4.

Table 4: Plume identification matrix.

Scenario	True-colour scene	MBMP/SWIR scene	CH_4 Plume?
A	Plume visible	No plume visible	No
B	No plume visible.	Bright four-pointed star like diffraction spike	No
C	No plume visible.	Plume visible with four-pointed diffraction spike.	Yes
D	No plume visible.	Plume visible	Yes

To tag the plumes, click anywhere on a plume in the map displayed by code 9 (fig. 105), the longitude and latitude coordinates will appear. Copy and paste each of these into code box 10 like so:

(31.6887, 5.8102), # User 1 (latitude, longitude)

(31.7910, 5.8263), # User 2 (latitude, longitude)

Additional lines for more plumes can be added as needed.

```
plume_coords = [
    (31.6584, 5.9054),  # Site 1 DO NOT DELETE OR MODIFY THESE!
    (31.6174, 5.9671),  # Site 2
    (31.7419, 5.8949),  # Site 3
    (31.7570, 5.9423),  # Site 4
    (31.7341, 5.9670),  # Site 5
    (31.7684, 5.9992),  # Site 6
    (31.7777, 5.9957),  # Site 7
    (31.7975, 6.0109),  # Site 8
    (31.7570, 6.1692),  # Site 9
    (31.8054, 6.1551),  # Site 10
    (31.8640, 6.1733),  # Site 11

    # User inputted plumes go below this message. Add more lines as needed
    (31.8926, 6.0389),  # User plume 1
]
```

Code 10: plume coordinate tags

Regression Model Development

A regression model is a statistical tool used to predict a dependent variable (here, methane emission rate in kg/h) based on independent variables. It works by identifying relationships in the training data and using these to estimate outcomes for new data.

To train the model, data from methane plumes with emission rates documented in peer-reviewed studies was collected (Gorroño et al., 2023; Pandey et al., 2023; Naus et al., 2023; Varon et al., 2021; Wang et al., 2023; Sanchez-Garcia et al., 2021). These plumes were then found using the MBMP Plume Visualiser. Each plume was measured for:

- **CS-Sum**: The plume intensity in its cross-section after subtracting background values.
- **Plume Length**: The plume's length in pixels.
- **Wind Speed**: ERA5 reanalysis data for the wind speed at the time of observation.

The regression analysis identifies how these factors relate to emission rates, allowing the model to predict methane emissions for other plumes based on their characteristics.

Below are the data that was collected for the regression analysis. The data used for the model as of publication, is listed below.

Table 5: Plume measurements found in peer reviewed journals used as training data.

Date	Lat, Long	Q (kg/h)	CS-Sum	Length (pixels)	Width (pixels)	Wind (m/s)	Platform	Source
2021-08-31	31.8066, 6.1545	5,453	0.066746	16.125	7.0	4.45	Sentinel-2	Gorroño et al., 2023
2020-01-04	31.8647, 6.1736	21,000	0.574297	294.544	47.0	3.65	Sentinel-2	Pandey et al., 2023
2019-11-20	31.6585, 5.9053	8,497	0.682773	106.231	38.0	0.49	Sentinel-2	Varon et al., 2021
2021-08-19	31.769, 6.0015	4,326	0.137231	43.174	13.0	0.96	PRISMA	Sanchez-Garcia et al., 2021
2021-08-19	31.7789, 5.9952	2,160	0.095425	13.601	11.0	0.96	PRISMA	Sanchez-Garcia et al., 2021
2021-08-19	31.7981, 6.0107	2,757	0.100792	16.124	8.0	0.96	PRISMA	Sanchez-Garcia et al., 2021
2020-01-07	31.659, 5.9055	8,240	0.692199	164.125	64.0	1.44	Sentinel-2	Radman et al. 2023
2023-01-31	31.7775, 5.9954	3,400	0.124114	18.788	8.0	2.3	ISS EMI	Carbon Mapper Website
2024-09-29	31.7772, 5.9934	3,000	0.071961	20.125	7.0	8.88	ISS EMI	Carbon Mapper Website
2020-01-14	31.7571, 6.1684	3,700	0.33289	53.460	30.0	1.92	Sentinel-2	Naus et al. 2023
2020-01-02	31.6172, 5.9674	3,600	0.206687	38.013	13.0	1.33	Sentinel-2	Naus et al. 2023
2020-08-06	31.7776, 5.9917	4,800	0.070314	18.439	8.0	5.66	Sentinel-2	Naus et al. 2023
2020-08-14	31.7692, 5.9987	3,400	0.100685	21.541	13.0	5.13	Sentinel-2	Naus et al. 2023
2020-02-28	31.7341, 5.9677	2,700	0.275828	60.745	30.0	0.22	Sentinel-2	Naus et al. 2023
2020-02-28	31.7569, 5.9422	2,100	0.067852	8.485	6.0	0.22	Sentinel-2	Naus et al. 2023
2020-07-30	31.6591, 5.9044	14,800	0.430201	72.173	35.0	5.51	Sentinel-2	Naus et al. 2023

Code box 11 allows for more example plumes to be added to improve the model, should more studies become available.

```
initial_data = {
    "Emission_rate_kg_h": [5453, 21000, 8497, 4326, 2160, 2757, 8240, 3400, 3000,
                           3700, 3600, 4800, 3400, 2700, 2100, 14800],
    "Cross_sectional_Adjusted_Sum": [0.06575300, 0.55398000, 0.64867300, 0.12791300, 0.09438400,
                                     0.09406700, 0.77022700, 0.12387600, 0.08018100, 0.27279700,
                                     0.20713000, 0.06979700, 0.12493400, 0.43429500, 0.06779200,
                                     0.08860200],
    "Plume_length": [16.125, 119.670, 102.591, 43.174, 13.601, 14.422, 148.762, 18.788, 7.071,
                     53.460, 36.878, 20.396, 25.318, 89.185, 8.485, 23.195],
    "Width": [7.000, 23.000, 32.000, 13.000, 11.000, 8.000, 53.000, 8.000, 6.000,
```

```
                19.000, 13.000, 8.000, 15.000, 39.000, 6.000, 8.000],
    "Wind_speed": [4.45, 3.65, 0.49, 0.96, 0.96, 0.96, 1.44, 2.3, 8.88, 1.92, 1.33,
                   5.66, 5.13, 0.22, 0.22, 5.51]
}
```

Code 11: training data for regression model.

Determining Wind Speed

Wind speed is a crucial factor in determining emission rate. Code box 12 determines the wind speed on the "Active Emission" date as an input for part of the regression analysis. Several warning messages will appear but these can be ignored.

Remember: as shown in section 3.3.9 Installing CDS API, the first time this code box is run, a "403 Client error: Forbidden for URL" error will appear. Please refer to section 3.3.9 for instructions on proceeding.

```
""" The first function calculates a centroid for the wind speed location
using the bounding box of the study area as the ERA5 API requires a
point location.
"""
def get_location_from_site_id(site_id, csv_path):
    df = pd.read_csv(csv_path)
    site = df[df['id'] == site_id]
    site = site.iloc[0]
    center_lat = (site['south'] + site['north']) / 2
    center_lon = (site['west'] + site['east']) / 2
    return {'latitude': center_lat, 'longitude': center_lon}

# Get the location for the ERA5 data request
location = get_location_from_site_id(site_id, csv_path)

"""
The cdsapi needs to be set up as per the instructions in the How
to Guide or this will not work!
"""
c = cdsapi.Client()

date = active_temporal_extent[0]  # this takes the date to be the same as the
active_emission function.

"""
This tool is hard coded to retrieve data for 10:00am as Sentinel-2
overpasses occur at around 10:30am If this tool is reconfigured for
another region of the world this may need to be adjusted.
"""
# Retrieve ERA5 data and store it in a temporary file
with NamedTemporaryFile(suffix='.nc') as tmp_file:
    result = c.retrieve(
        'reanalysis-era5-single-levels',
        {
            'product_type': 'reanalysis',
            'variable': ['10m_u_component_of_wind', '10m_v_component_of_wind'],
            'year': date.split('-')[0],
            'month': date.split('-')[1],
            'day': date.split('-')[2],
            'time': ['10:00'],  # Sentinel 2 overpasses are at around 10:30 am over
Algeria.
            'format': 'netcdf',  # NetCDF format
            'area': [
                location['latitude'] + 0.25, location['longitude'] - 0.25,
                location['latitude'] - 0.25, location['longitude'] + 0.25,
            ],
```

```
        }
    )
    # Download data to the temporary file
    result.download(tmp_file.name)

    # Load the dataset with xarray
    ds = xr.open_dataset(tmp_file.name)

"""
ERA5 data provides wind speed in east/west (u10) and north/south (v10).
Positive u10 and v10 equals an east and north wind. Negative values are
the reverse.
"""
# Extract u and v components
u10 = ds['u10'].sel(latitude=location['latitude'], longitude=location['longitude'],
method='nearest')
v10 = ds['v10'].sel(latitude=location['latitude'], longitude=location['longitude'],
method='nearest')

"""
u10 and v10 form two sides of a right-angled triangle so we can calculate
the wind speed using the A^2 + B^2 = C^2 (Pythagoras, 530 BCE). With the
wind variables this would be: u10^2 + v10^2 = windspeed^2. So this can be
reconfigured as wind speed = the square root of (u10² + v10²).
"""
# Wind speed calculation
wind_speed = np.sqrt(u10**2 + v10**2)

# Extract wind speed value
wind_speed_value = wind_speed.values.item()
print(f"Wind Speed at 10:00 on {date}: {wind_speed_value:.2f} m/s")
```

Code 12: code for wind speed.

Running the tagged plume analysis

Code box 13 analyses methane plumes tagged earlier and provides the following information:

- Plume Insights: Locations, sizes, and predicted methane emission rates (kg/h) visualised on an interactive map and summarised in a table.
- Model Evaluation: Details on the regression model used to estimate emissions, including its performance metrics (e.g., R^2 and MSE).
- Interactive Visualisation: A map with SWIR data overlays, plume boundaries, and tooltips for detailed exploration.

```
swir_diff_path = r'C:\GIS_Course\Methane_Point_Detection\Sentinel-
2_Algeria_Methane\SWIR_diff_masked_urban.tiff'

# Function to open the swir_diff_path file for analysis.
with rasterio.open(swir_diff_path) as tiff_file:
    raster_data = tiff_file.read(1)  # Read the first band
    bounds = tiff_file.bounds
    transform = tiff_file.transform

    # Define nodata_value properly
    nodata_value = tiff_file.nodata  # Extract from metadata if available
    if nodata_value is None:
        nodata_value = -32768

# Converts raster data into a masked array and hides the no data pixels.
masked_data = np.ma.masked_equal(raster_data, nodata_value)

# Compute min and max from remaining pixels.
lower_bound = masked_data.min()
upper_bound = masked_data.max()
normalized_data = (masked_data - lower_bound) / (upper_bound - lower_bound)

# function to centre the folium map on the raster
def get_raster_centre(tiff_path):
    with rasterio.open(tiff_path) as tiff_file:
        bounds = tiff_file.bounds
        centre_lat = (bounds.top + bounds.bottom) / 2
        centre_lon = (bounds.left + bounds.right) / 2
    return centre_lat, centre_lon

"""
This function sets up a bresenham algorithm to find all the pixel
coordinates that form a straight line between two points.

    x0, y0: Start point coordinates.
    x1, y1: End point coordinates.

It then records the values of each of those pixels.
```

```
"""
def bresenham_line(x0, y0, x1, y1):
    points = []  # Stores the pixels forming the line
    dx = abs(x1 - x0)
    dy = abs(y1 - y0)

    # Determine movement direction (+1 or -1 for each axis)
    step_x = 1 if x0 < x1 else -1
    step_y = 1 if y0 < y1 else -1

    # keeps track of how far the current point is from the ideal straight line
    err = dx - dy

    while (x0, y0) != (x1, y1):  # Stop when reaching the endpoint
        points.append((x0, y0))  # Store the current position's value

        # double_err is used to decide whether to move horizontally, vertically, or
diagonally.
        double_err = err * 2 #
        if double_err > -dy:
            err -= dy
            x0 += step_x
        if double_err < dx:
            err += dx
            y0 += step_y

    points.append((x1, y1))
    return points

# This function takes the bresenham_line and then applies it to the masked dataset.
def get_line_pixel_values(start, end, masked_data):
    line_pixels = bresenham_line(int(start[0]), int(start[1]), int(end[0]), int(end[1]))
    pixel_values = [masked_data[row, col] for row, col in line_pixels if 0 <= row <
masked_data.shape[0] and 0 <= col < masked_data.shape[1]]
    return pixel_values, line_pixels

""" calculate_plume_dimensions determines:
    - plume_length: The longest distance between any two points in the plume (float).
    - plume_width: The perpendicular width of the plume using PCA (float)."""
def calculate_plume_dimensions(plume_pixels):
    # Compute the convex hull
    hull = ConvexHull(plume_pixels)
    hull_points = plume_pixels[hull.vertices]

    # Compute the plume length as the maximum pairwise distance between hull points
    plume_length = pdist(hull_points).max()

    # Use PCA to determine the major axis of the plume
```

```
    pca = PCA(n_components=2)
    pca.fit(plume_pixels)
    main_direction = pca.components_[0]
    perp_direction = np.array([-main_direction[1], main_direction[0]])  # Perpendicular
vector

    # Project plume pixels onto the perpendicular vector to compute width
    projections = plume_pixels @ perp_direction
    plume_width = projections.max() - projections.min()

    return plume_length, plume_width

""" analyze_plume_with_cross_section_sum performs the main plume analysis. For each
tagged plume it will return its:
    - C/S Sum
    - Length
    - Width
    - emission rate (Q)

It also creates the folium map and a dataframe to show the data"""
def analyze_plume_with_cross_section_sum(masked_data, plume_coords, transform,
initial_centre):
    # Folium map perameters
    plume_map = folium.Map(location=initial_centre, zoom_start=11, control_scale=True)

    plume_results = [] # A place to store the plume results.

    """Labeled_array identifies plume regions using an absolute SWIR threshold
    (`absolute_threshold`). This ensures consistent detection across different
    scenes, independent of pixel distribution. Adjacent pixels are then grouped
    and labeled as individual plumes."""
    absolute_threshold = 0.009  # pixels at or above this value are marked as plumes
    labeled_array, _ = label(masked_data > absolute_threshold)

    # Loops through each suspected plume location.
    for i, (lat, lon) in enumerate(plume_coords):
        try:
            # Convert Lat/Long to raster grid coordinates.
            row, col = rasterio.transform.rowcol(transform, lon, lat)
            row, col = int(row), int(col)

            # This defines a 50 pixel by 50 pixel search box around the plume coordinate
            window_size = 50
            half_window = window_size // 2

            # Get the boundary of the search area in row/col coordinates
            row_start, row_end = max(0, row - half_window), min(masked_data.shape[0],
row + half_window + 1)
```

```
            col_start, col_end = max(0, col - half_window), min(masked_data.shape[1],
col + half_window + 1)

            # Check if a plume is within the search box.
            plume_label = labeled_array[row, col]

            # Extract a local window of pixels around (row, col)
            row_start, row_end = max(0, row - half_window), min(masked_data.shape[0],
row + half_window + 1)
            col_start, col_end = max(0, col - half_window), min(masked_data.shape[1],
col + half_window + 1)
            local_window = labeled_array[row_start:row_end, col_start:col_end]

            """Because of the slightly noisy quality of the data, individual pixels
might
            trigger the plume detection code. To minimse the risk of this the following
code
            specifies that 20 adjacent plume pixels are required to be counted as a
plume. """

            # Identify groups of connected plume pixels
            binary_window = (local_window > 0).astype(int)  # Convert plume labels to
binary (1 = plume, 0 = no plume)
            labeled_clusters, num_clusters = label(binary_window)  # Label connected
components

            # Find the largest connected cluster size in the search box
            cluster_sizes = np.bincount(labeled_clusters.ravel())[1:]  # Ignore
background (index 0)
            largest_cluster_size = cluster_sizes.max() if len(cluster_sizes) > 0 else 0

            # Set a minimum threshold for a valid plume detection
            min_cluster_size = 20  # This could be changed to a higher value if false
postives persist.

            if largest_cluster_size >= min_cluster_size:
                # in the event of an overlapping plume, this selects the one with the
largest pixel area.
                # the smaller plume will need to be tagged by the user
                unique_labels, counts = np.unique(local_window[local_window > 0],
return_counts=True)
                plume_label = unique_labels[np.argmax(counts)] if len(unique_labels) > 0
else 0
            else:
                plume_label = 0  # No valid plume detected

            if plume_label == 0:
```

```
                plume_results.append({"Plume": i + 1,"Location": (lat, lon), "Status":
"No plume"})
                continue

            # Extract plume pixels.
            plume_region = labeled_array == plume_label
            plume_pixels = np.column_stack(np.where(plume_region))

            # Compute plume width & legnth.
            plume_length, plume_width = calculate_plume_dimensions(plume_pixels)

            # Compute the plume centroid to serve as the cross section point.
            centroid = plume_pixels.mean(axis=0)

            # Draw a perpendicular cross-section line.
            pca = PCA(n_components=2)
            pca.fit(plume_pixels)
            perp_direction = [-pca.components_[0, 1], pca.components_[0, 0]]

            # Define the perpendicular line in pixel coordinates.
            perp_line_coords = [
                (centroid[0] - perp_direction[0] * plume_width / 2, centroid[1] -
perp_direction[1] * plume_width / 2),
                (centroid[0] + perp_direction[0] * plume_width / 2, centroid[1] +
perp_direction[1] * plume_width / 2),
            ]

            # Convert line coordinates to latitude/longitude for mapping
            perp_line_latlon = [
                rasterio.transform.xy(transform, int(pt[0]), int(pt[1])) for pt in
perp_line_coords
            ]

            # Add the cross-sectional line to the Folium map
            folium.PolyLine(
                locations=[(lat, lon) for lon, lat in perp_line_latlon],
                color="blue",
                weight=2,
                popup=f"Cross Section for Plume {i + 1}"
            ).add_to(plume_map)

            # Get pixel values along the perpendicular line.
            line_pixel_values, line_pixels = get_line_pixel_values(perp_line_coords[0],
perp_line_coords[1], masked_data)
            num_intersecting_pixels = len(line_pixels)

            # Compute cross-sectional sum.
            adjusted_sum = sum(abs(value) for value in line_pixel_values)
```

```
            # Create a convex hull to outline the plume for visualisation.
            hull = ConvexHull(plume_pixels)
            hull_coords = [(plume_pixels[vertex][0], plume_pixels[vertex][1]) for vertex
in hull.vertices]
            hull_latlon = [rasterio.transform.xy(transform, int(pt[0]), int(pt[1])) for
pt in hull_coords]

            # Add the plume outline on the map with the site label
            site_label = f"Site {i + 1}" if i < 11 else f"User {i - 10}"
            folium.Polygon(
                locations=[(lat, lon) for lon, lat in hull_latlon],
                color="red",
                weight=3,
                fill=True,
                fill_opacity=0.2,
                popup=site_label,  # Use site label instead of "Plume X"
            ).add_to(plume_map)

            plume_results.append({
                "Plume": i + 1,
                "Location": (lat, lon),
                "Width (px)": num_intersecting_pixels if plume_label > 0 else 0,
                "C/S Sum": adjusted_sum if plume_label > 0 else 0,
                "Length (px)": plume_length if plume_label > 0 else 0,
                "Q (kg/h)": None,  # Placeholder for emission rate
                "Status": "Detected" if plume_label > 0 else "No plume",
            })


        # If a plume measurement encounters an error, this will allow any other plumes
to be measured.
        except Exception as e:
            plume_results.append({"Plume": i + 1, "Location": (lat, lon), "Status":
f"Error: {e}"})

    # Return measured plume stats and folium map.
    return plume_results, plume_map

""" add_swir_data_to_map loads, normalizes, and adds a Short-Wave
Infrared (SWIR) raster layer to an interactive Folium map."""
def add_swir_data_to_map(map_object, tiff_path):
    with rasterio.open(tiff_path) as tiff_file:
        swir_data = tiff_file.read(1)
        bounds = tiff_file.bounds
        nodata_value = -32768.0  # NaN value to be ignored.

    # Mask NaN pixels and normalise data.
```

```
    masked_data = np.ma.masked_equal(swir_data, nodata_value)
    filtered_masked_data = masked_data[masked_data >= -3000]  # Ignore values below -
3000
    mean, std = np.nanmean(filtered_masked_data), np.nanstd(filtered_masked_data)
    lower_bound, upper_bound = mean - std_factor * std, mean + std_factor * std
    normalized_data = (masked_data - lower_bound) / (upper_bound - lower_bound)
    normalized_data = np.clip(normalized_data, 0, 1)

    # Convert SWIR data to RGB for mapping.
    cmap = plt.get_cmap("viridis")
    swir_rgb = (cmap(normalized_data)[:, :, :3] * 255).astype(np.uint8)
    image_bounds = [[bounds.bottom, bounds.left], [bounds.top, bounds.right]]

    # Add SWIR overlay to the map.
    swir_overlay = ImageOverlay(
        name="SWIR Data",
        image=swir_rgb,
        bounds=image_bounds,
        opacity=1,  # Match Truecolour opacity
        interactive=True,  # Allow user interaction
        zindex=2  # Place above Truecolour
    )
    swir_overlay.add_to(map_object)

# Convert the dataset into a DataFrame
model_df = pd.DataFrame(initial_data)

""" Update_model trains and evaluates a XGBoost regression model
to predict methane emission rates based on the known plume data shown
earlier."""

def update_model(df):
    # Extract independent variables (X) and dependent variable (y)
    X = df[["Cross_sectional_Adjusted_Sum", "Wind_speed", "Plume_length", "Width"]]
    y = df["Emission_rate_kg_h"]

    # Apply log transformation to y correctly
    y_log = np.log(y + 1)  # Adding 1 to prevent log(0) errors

    # Standardize the features properly
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    # Train XGBoost model with optimized settings
    xgb_model = XGBRegressor(
        objective="reg:squarederror",
        n_estimators=500,
        learning_rate=0.01,
```

```
        max_depth=5,
        colsample_bytree=0.8,
        subsample=0.8,
        gamma=0.1,
        reg_alpha=0.1,
        reg_lambda=0.1,
        random_state=42 # what is the meaning of Life the Universe and Everything?
    )

    # Train the model on the log-transformed y
    xgb_model.fit(X_scaled, y_log)

    return X_scaled, y, scaler, xgb_model

# Train the improved XGBoost model
X_scaled, y, scaler, xgb_model = update_model(model_df)

# Get the centre of the SWIR TIFF
centre_coords = get_raster_centre(swir_diff_path)

# Perform plume analysis, centreing the map on the SWIR TIFF
plume_analysis_results, plume_map = analyze_plume_with_cross_section_sum(masked_data,
plume_coords, transform, centre_coords)

# Add wind speed to each plume analysis result
for plume in plume_analysis_results:
    if plume["Status"] == "Detected":  # Only predict for detected plumes
        adjusted_sum = plume["C/S Sum"]
        plume_length = plume["Length (px)"]
        width_value = plume["Width (px)"]  # Extract width from the plume data

        # Convert input features to a DataFrame to maintain feature names
        input_features = pd.DataFrame([[adjusted_sum, wind_speed_value, plume_length,
width_value]],
                                      columns=["Cross_sectional_Adjusted_Sum",
"Wind_speed", "Plume_length", "Width"])
        # Scale the input data using the same scaler
        input_features_scaled = scaler.transform(input_features)
        # Use the trained XGBoost model for prediction
        log_prediction = xgb_model.predict(input_features_scaled)
        # Convert back from log scale
        emission_rate = np.exp(log_prediction[0]) - 1
        # Store the emission rate
        plume["Predicted Emission Rate (kg/h)"] = emission_rate

# Convert updated results to a DataFrame
plume_df = pd.DataFrame(plume_analysis_results)
# Rename the "Plume" index to "Site X" or "User X"
```

```
plume_labels = [f"Site {i + 1}" if i < 11 else f"User {i - 10}" for i in
range(len(plume_df))]

# Set the custom labels as the new index
plume_df.index = plume_labels

plume_df = plume_df.rename(columns={
    "Cross_sectional_Adjusted_Sum": "C/S Sum",
    "Plume_length": "Length (px)",
    "Width": "Width (px)",
    "Predicted Emission Rate (kg/h)": "Q (kg/h)"
})

# Dataframe column order
column_order = ["Status", "Location", "Width (px)", "C/S Sum", "Length (px)", "Q
(kg/h)"]
existing_columns = [col for col in column_order if col in plume_df.columns]
plume_df = plume_df[existing_columns]

# Hiding the dataframe keyfield as it isn't needed
if "Plume" in plume_df.columns: plume_df.drop(columns=["Plume"], inplace=True)

# Display updated DataFrame with predicted emission rates
r_squared = xgb_model.score(X_scaled, np.log(model_df["Emission_rate_kg_h"] + 1))
print(f"Model R-squared (R²): {r_squared:.4f}")
print(f"Active Emission Date: {active_temporal_extent[0]}")
print(f"No Emission Date: {no_temporal_extent[0]}")
print(f"Wind Speed: {wind_speed_value:.2f} m/s")
pd.set_option("display.width", 200)  #Controls dataframe width
pd.set_option("display.max_columns", None)  # "None" puts all a plume's results on one
line
# Replace NaN values with an empty string for readability
plume_df = plume_df.fillna("")

# Display the updated DataFrame
print(plume_df)

# Load the true colour image
truecolour_sat = 'Sentinel-2_truecolour_reprojected.Tiff'
img = rasterio.open(truecolour_sat)
blue, green, red = img.read(1), img.read(2), img.read(3)

# Adjust brightness of truecolour image
brightness_factor = 0.03 # only change this if the truecolour image is too dark or
bright
blue = np.clip(blue * brightness_factor, 0, 255)
green = np.clip(green * brightness_factor, 0, 255)
red = np.clip(red * brightness_factor, 0, 255)
```

```
# Stack bands to create RGB image
rgb = np.dstack((red, green, blue))
rgb = rgb / rgb.max()
rgb = np.log1p(rgb)
rgb = rgb / rgb.max()

# Add true colour image overlay
truecolour_overlay = ImageOverlay(
    name= "Truecolour",
    image=rgb,
    bounds=swir_bounds,
    opacity=1,  # Lower opacity for blending with SWIR overlay
    interactive=True,
    zindex=1,
)
truecolour_overlay.add_to(plume_map)

# Add SWIR overlay to the map
add_swir_data_to_map(plume_map, swir_diff_path)

# Add a layer control to toggle map layers
LayerControl().add_to(plume_map)

# Display the map with updated analysis
display(plume_map)
```

Code 13: code for tagged plume analysis.

Once code 13 is run the analysis is complete and the information shown in figure 107 will be shown, including the estimated emission rates for the tagged plumes in kg/h.

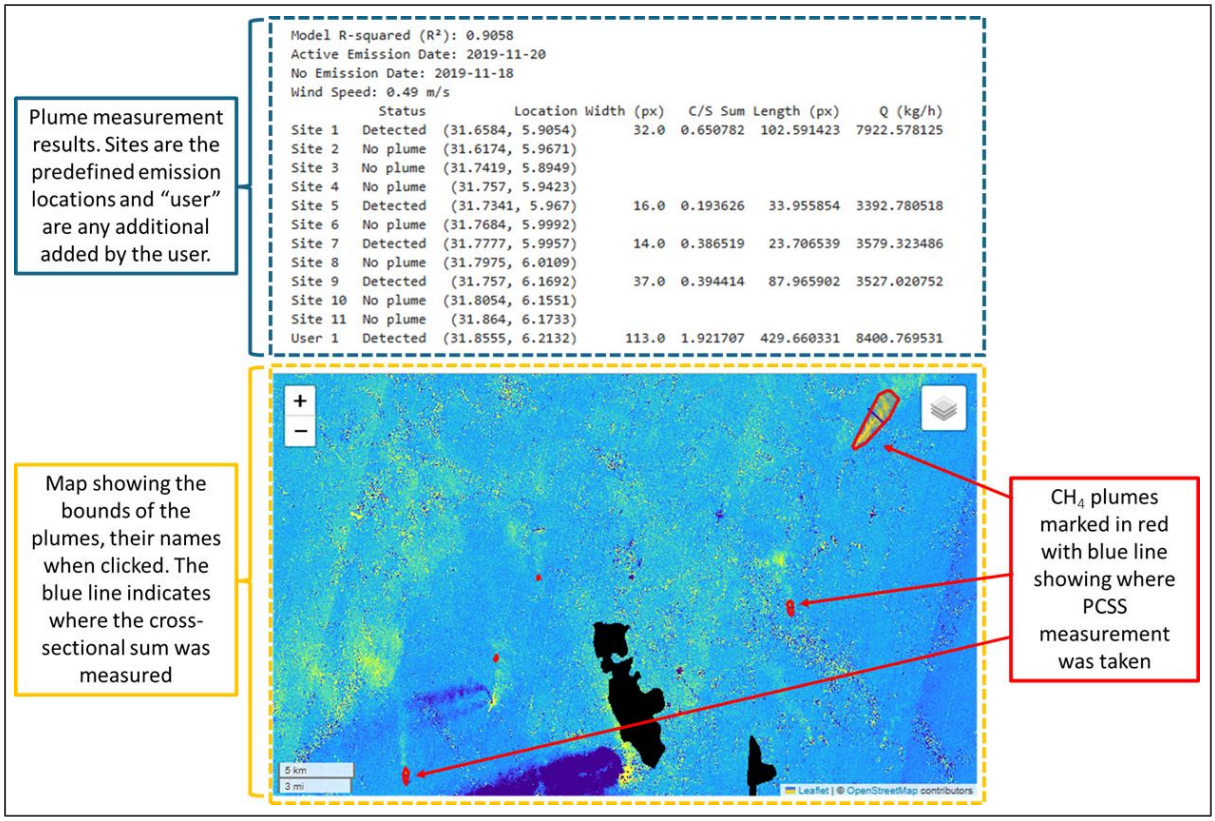

Figure 107: Results of plume analysis and model performance data.

3.5 Troubleshooting

Input errors have been covered in the tool notebooks, however there are errors unrelated to a user input which can cause problems. These are detailed below.

3.5.1 Remote disconnected error

```
Error: Remote disconnected
```

Or

```
OpenEoApiError: [500] Internal: Server error: KazooTimeoutError('Connection time-out') (ref: r-2405108830e742b59ef8ac2f28647fb5)
```

This can occur when there are issues with the Copernicus network. In the event that you see an error like this you can check page https://dataspace.copernicus.eu/news for any downtime messages and you can also contact the Copernicus dataspace team via the form at https://helpcenter.dataspace.copernicus.eu/hc/en-gb/requests/new

4. References

Abada, Z. and Bouharkat, M., 2018. Study of management strategy of energy resources in Algeria. *Energy Reports*, 4, pp.1-7. Available at: https://www.researchgate.net/publication/328663133_Study_of_management_strategy_of_energy_resources_in_Algeria [Accessed 18 Sep. 2024].

Dowd, E., Manning, A.J., Orth-Lashley, B., Girard, M., France, J., Fisher, R.E., Lowry, D., Lanoisellé, M., Pitt, J.R., Stanley, K.M. and O'Doherty, S., 2023. First validation of high-resolution satellite-derived methane emissions from an active gas leak in the UK. *EGUsphere*, 2023, pp.1-22.

Elkind, J., Blanton, E., Denier Van Der Gon, H., Kleinberg, R.L. and Leemhuis, A., 2020. Nowhere to hide: The implications of satellite-based methane detection for policy, industry and finance. *Columbia Center on Global Energy Policy*. Available at: https://energypolicy.columbia.edu

European Commission, 2023. EU announces €175m financial support to reduce methane emissions at COP28. *IP/23/6057*. Available at: https://ec.europa.eu/commission/presscorner/detail/en/IP_23_6057

Ferronato, N., Torretta, V., Ragazzi, M., & Rada, E.C. (2017). Waste mismanagement in developing countries: A case study of environmental contamination. UPB Sci. Bull, 79(2), 185-196.

Google Earth. (2024). Satellite view of Algerian petrochemical facility. Retrieved 10th of May 2024.

Integrated Methane Inversion (2024) *IMI processing steps*, Harvard University. Available at: https://imi.seas.harvard.edu/overview (Accessed: 27 November 2024).

International Energy Agency, 2022. Global Methane Tracker 2022. *IEA, Paris*. Available at: https://www.iea.org/reports/global-methane-tracker-2022

International Energy Agency, 2024. Global Methane Tracker 2024: Tracking pledges, targets, and action. Available at: https://www.iea.org/reports/global-methane-tracker-2024/tracking-pledges-targets-and-action [Accessed 17 Sep. 2024].

Jet Propulsion Laboratory. (2024). VISIONS: The EMIT Open Data Portal. Retrieved 10th of May 2024 from: https://tinyurl.com/2s4kkwe5

Jackson, R.B., Saunois, M., Bousquet, P., Canadell, J.G., Poulter, B., Stavert, A.R., Bergamaschi, P., Niwa, Y., Segers, A., and Tsuruta, A., 2020. Increasing anthropogenic methane emissions arise equally from agricultural and fossil fuel sources. *Environmental Research Letters*, 15(7), p.071002. Available at: https://doi.org/10.1088/1748-9326/ab9ed2 [Accessed 4 Jul. 2024].

Karimi, N., Ng, K.T.W. and Richter, A., 2021. Prediction of fugitive landfill gas hotspots using a random forest algorithm and Sentinel-2 data. *Sustainable Cities and Society*, 73, p.103097.

Liu, M., Van Der A, R., Van Weele, M., Eskes, H., Lu, X., Veefkind, P., De Laat, J., Kong, H., Wang, J., Sun, J. and Ding, J., 2021. A new divergence method to quantify methane emissions using observations of Sentinel-5P TROPOMI. *Geophysical Research Letters*, *48*(18), p.e2021GL094151.

Lorente, A., Borsdorff, T., Butz, A., Hasekamp, O., Aan De Brugh, J., Schneider, A., Wu, L., Hase, F., Kivi, R., Wunch, D. and Pollard, D.F., 2021. Methane retrieved from TROPOMI: Improvement of the data product and validation of the first 2 years of measurements. *Atmospheric Measurement Techniques*, *14*(1), pp.665-684.

Naus, S., Maasakkers, J.D., Gautam, R., Omara, M., Stikker, R., Veenstra, A.K., Nathan, B., Irakulis-Loitxate, I., Guanter, L., Pandey, S. and Girard, M., 2023. Assessing the relative importance of satellite-

detected methane superemitters in quantifying total emissions for oil and gas production areas in Algeria. *Environmental Science & Technology*, 57(48), pp.19545-19556.

Malley, C.S., Borgford-Parnell, N., Haeussling, S., Howard, I.C., Lefèvre, E.N. and Kuylenstierna, J.C., 2023. A roadmap to achieve the global methane pledge. *Environmental Research: Climate*, 2(1), p.011003.

McNabb, R. (2024). Reprojecting Rasters Using Rasterio [Blog post]. Retrieved 24th of April 2024 from https://iamdonovan.github.io/teaching/egm722/practicals/raster.html

Microsoft Copilot Designer. (2024). Cover artwork. Generated on 25th of November 2024

OpenEO. (2024). NDVI Timeseries. [Online]. Retrieved 22nd of April 2024, from: https://documentation.dataspace.copernicus.eu/notebook-samples/openeo/NDVI_Timeseries.html

Pandey, S., van Nistelrooij, M., Maasakkers, J.D., Sutar, P., Houweling, S., Varon, D.J., Tol, P., Gains, D., Worden, J., & Aben, I. (2023). Daily detection and quantification of methane leaks using Sentinel-3: a tiered satellite observation approach with Sentinel-2 and Sentinel-5p. Remote Sensing of Environment, 296, 113716.

Parker, R., Boesch, H., Cogan, A., Fraser, A., Feng, L., Palmer, P.I., Messerschmidt, J., Deutscher, N., Griffith, D.W., Notholt, J., & Wennberg, P.O. (2011). Methane observations from the Greenhouse Gases Observing SATellite: Comparison to ground based TCCON data and model calculations. Geophysical Research Letters, 38(15).

Rosenthal, J., 2023. In Reverse: Natural Gas and Politics in the Maghreb and Europe. *FPRI: Foreign Policy Research Institute*. United States of America. Available at: https://coilink.org/20.500.12592/bf2q0b [Accessed 29 September 2024].

Sentinel Hub. (2024) "About Sentinel-2 Data." Retrieved 5th of May 2024, from: https://docs.sentinel-hub.com/api/latest/data/sentinel-2-l2a/

Sentinel Hub. (2024) "About Sentinel-5P Data." Retrieved 23rd April 2024, from: https://docs.sentinel-hub.com/api/latest/data/sentinel-5p-l2/

Shen, L., Gautam, R., Omara, M., Zavala-Araiza, D., Maasakkers, J.D., Scarpelli, T.R., Lorente, A., Lyon, D., Sheng, J., Varon, D.J. and Nesser, H., 2022. Satellite quantification of oil and natural gas methane emissions in the US and Canada including contributions from individual basins. Atmospheric Chemistry and Physics, 22(17), pp.11203-11215.

Sherwin, E.D., Rutherford, J.S., Zhang, Z., Chen, Y., Wetherley, E.B., Yakovlev, P.V., Berman, E.S., Jones, B.B., Cusworth, D.H., Thorpe, A.K. and Ayasse, A.K., 2024. US oil and gas system emissions from nearly one million aerial site measurements. *Nature*, *627*(8003), pp.328-334.

Sonneveld, E. (2024). Obtaining a list of available Sentinel 2 datasets for a location. Copernicus Dataspace Forum. Retrieved 25th of April 2024 from https://tinyurl.com/4t2trevv

Talamali, S., Chaouchi, R. and Benayad, S., 2016. Sedimentological evolution of the Lower Series formation in the southern area of the Hassi R'Mel field, Saharan Platform, Algeria. *Arabian Journal of Geosciences*, 9, p.481. DOI: 10.1007/s12517-016-2506-7.

United Nations Development Programme, 2022. Country Programme Document for Algeria (2023-2027). DP/DCP/DZA/4. Available at: https://www.undp.org/sites/g/files/zskgke326/files/2023-03/CPD%20Algeria%202023-2027.pdf [Accessed 17 September 2024].

Varon, D.J., Jacob, D.J., McKeever, J., Jervis, D., Durak, B.O., Xia, Y. and Huang, Y., 2018. Quantifying methane point sources from fine-scale satellite observations of atmospheric methane plumes. *Atmospheric Measurement Techniques*, 11(10), pp.5673-5686.

Varon, D. J., Jervis, D., McKeever, J., Spence, I., Gains, D., & Jacob, D. J., 2021. High-frequency monitoring of anomalous methane point sources with multispectral Sentinel-2 satellite observations. Atmospheric Measurement Techniques, 14, 2771–2785.

Varon, D.J., Jacob, D.J., Sulprizio, M., Estrada, L.A., Downs, W.B., Shen, L., Hancock, S.E., Nesser, H., Qu, Z., Penn, E., Chen, Z., Lu, X., Lorente, A., Tewari, A. and Randles, C.A., 2022. Integrated Methane Inversion (IMI 1.0): a user-friendly, cloud-based facility for inferring high-resolution methane emissions from TROPOMI satellite observations. *Geoscientific Model Development*, 15, pp.5787–5805. https://doi.org/10.5194/gmd-15-5787-2022.

Vigano, I., Van Weelden, H., Holzinger, R., Keppler, F., McLeod, A., & Röckmann, T. (2008). Effect of UV radiation and temperature on the emission of methane from plant biomass and structural components. Biogeosciences, 5(3), 937-947.

Wang, H., Fan, X., Jian, H. and Yan, F., 2024. Exploiting the Matched Filter to Improve the Detection of Methane Plumes with Sentinel-2 Data. *Remote Sensing*, *16*(6), p.1023.