



UK Centre for
Ecology & Hydrology

Earth observation for malaria modelling: a practical toolkit for satellite-based prediction of mosquito distributions using Google Earth Engine and R

Date 08/08/2022

Christopher G. Marston¹, Clare S. Rowland¹, Aneurin W. O'Neil¹, Seth Irish², Francis Wat'senga³, Pilar Martin-Gallego⁴, Patrick Giraudoux⁵, Clare Strode⁴.

¹ UK Centre for Ecology & Hydrology, Library Avenue, Bailrigg, Lancaster. United Kingdom. LA14AP.

² U.S. President's Malaria Initiative and Centers for Disease Control and Prevention, 1600 Clifton Road NE, Atlanta, GA, 30329. USA.

³ Institut National de Recherche Biomédicale (INRB), Avenue de la Démocratie N° 5345, Kinshasa – Gombe, Democratic Republic of the Congo.

⁴ Edge Hill University, St Helens Road, Ormskirk, Lancashire L394QP, United Kingdom.

⁵ Department of Chrono-Environment, University of Bourgogne Franche-Comte/CNRS, La Bouloie, 25030 Besançon CEDEX, France



Dataset documentation

Version 1.0, 08/08/2022

Version	Date	Updates
1.0	08/08/2022	Original release

Contents

1. Introduction	3
2. Software installation and registration	5
2.1 Installing QGIS	5
2.2 Installing R	7
2.3 Installing RStudio	8
2.4 Registration for Google Earth Engine	10
2.4.1 Registration for Google Drive	10
2.4.2 Registration for Google Earth Engine	11
3. Study area	14
4. Dataset generation – QGIS	15
4.1 Creating a validation point shapefile	16
4.2 Creating a training polygon shapefile	18
4.3 To create the study area extent shapefile	23
5. Google Earth Engine - satellite data pre-processing	24
5.1 GEE interface / GUI	24
5.2 Uploading assets	25
5.3 GEE scripts	26
5.4 Importing assets	29
5.3 Description of the script	30
5.3.1 Functions and display parameters	30
5.3.2 Sentinel-1 SAR processing for land cover classification	32
5.3.3 Topographical data processing	36
5.3.4 Sentinel-2 imagery collection processing	37
5.3.5 Random forest land cover classification	39
5.3.6 Calculate proportional areas of individual land cover classes using moving window	43
5.3.7 Calculate 'distance to' rasters	44
5.3.8 Vegetation indices	45
5.3.9 CHIRPS data extraction and smoothing	45
5.3.10 Scale and convert data to integer, compile bands and export	47
5.3.11 Import mosquito sampling data and extract band data	50
5.4. Running the script	53
6. R – feature selection	55
6.1 Load required packages and set working directory	57

7.	Google Earth Engine - modelling	63
8.	Google Earth Engine - data visualisation	68
9.	Disclaimer	73
10.	Acknowledgements	74
11.	References	75
12.	Glossary	76

1. Introduction

This user-guide accompanies the publication Marston *et al.* (2023). It provides a user-friendly introduction to implementing the satellite-based analysis and random forest modelling **to identify the key bio-geographical variables that influence mosquito distributions and abundance**. It is intended to be a resource for users with limited prior knowledge of analysis of this nature and presents step-by-step instructions for users to perform **predictive modelling of mosquito distributions**; sample datasets and analysis scripts are provided.

This guide uses three software packages, **Google Earth Engine, R (with RStudio)** and **QGIS** for **data pre-processing, modelling and visualisation**, all of which are **cost-free for non-commercial use**. Scripts are provided to perform data processing and analysis in both Google Earth Engine (GEE) and R, and although these scripts are designed to automate the analysis to a large degree, they are currently optimised for the study area and time period used in the worked example (i.e. Lodja, Democratic Republic of the Congo, see page 14). It will require user input to adapt the scripts to different study areas and time-periods of interest, and this will also require suitable mosquito survey data with associated location coordinates (i.e. latitude and longitude decimal degrees) and dates to be provided by the user. Users are free to adapt and build on this to suit their own purposes.

This user manual includes the following sections:

2. Software installation and registration
3. Study area
4. Dataset generation – QGIS
5. GEE - satellite data pre-processing
6. R – feature selection
7. GEE - data modelling
8. GEE - data visualisation

This user handbook comes with the following files:

- ✓ R script
- ✓ CSV file containing mosquito data from Lodja, DRC (supplied as an asset in the GEE script)

- ✓ Three Google Earth Engine (GEE) scripts
- ✓ Land cover training data (supplied as an asset in the GEE script)

By following the instructions, the user will learn to apply the methods, but will also produce key files necessary for subsequent stages of the methods. These may be useful should the user wish to apply these methods to different datasets, areas and scenarios of their choosing. The data processing workflow will involve moving between software packages at different stages of analysis, with a broad overview of the main processing stages and the corresponding software used presented in Figure 1.

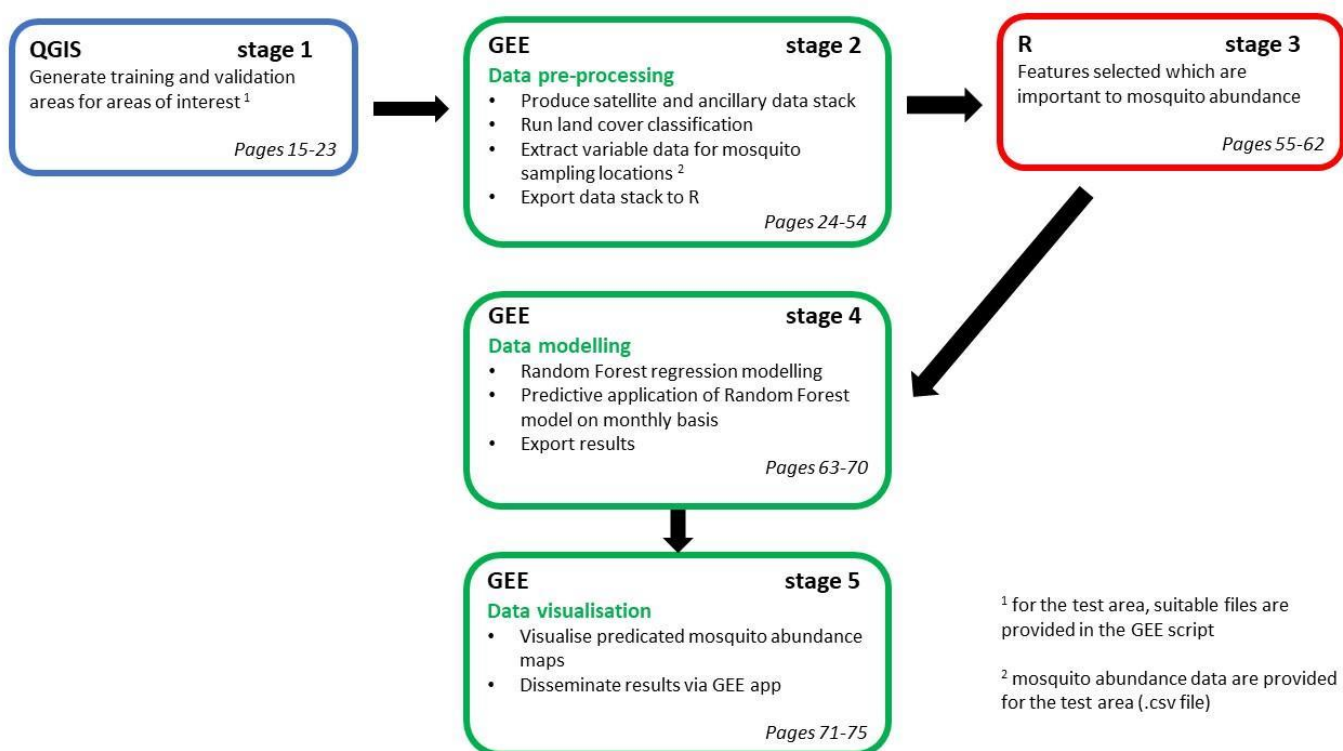


Figure 1. Broad overview of the main data processing stages and the corresponding software used

2. Software installation and registration

To implement the functionality in this user manual, it is necessary to install **QGIS** and **RStudio** on a local computer and register for a user account for **Google Earth Engine** for cloud-processing. The following documentation is for a 64-bit Windows operating system. Installation instructions for other operating systems may differ - in this case, please refer to the installation instructions for the respective software packages for your specific operating system. The processing and analysis contained in this guide required a significant amount of storage space. We would recommend ensuring at least **10gb** of free space is available prior to performing the analysis.

2.1 Installing QGIS

1. Go to the QGIS website – <https://www.qgis.org/en/site/> (Figure 2.1) and click on the ‘**Download Now**’ button.



Figure 2.1. QGIS Website

2. In the download window that opens (Figure 2.2), select the appropriate download option for the platform that you are using. Here, we will select the most UKCEH report ... version 1.0

recent QGIS standalone Installer Version 3.20 (64 bit). QGIS updates are released at regular updates, so the most recent version available may differ from the 3.20 version used here. Click the link '**QGIS Standalone Installer Version 3.20**'.

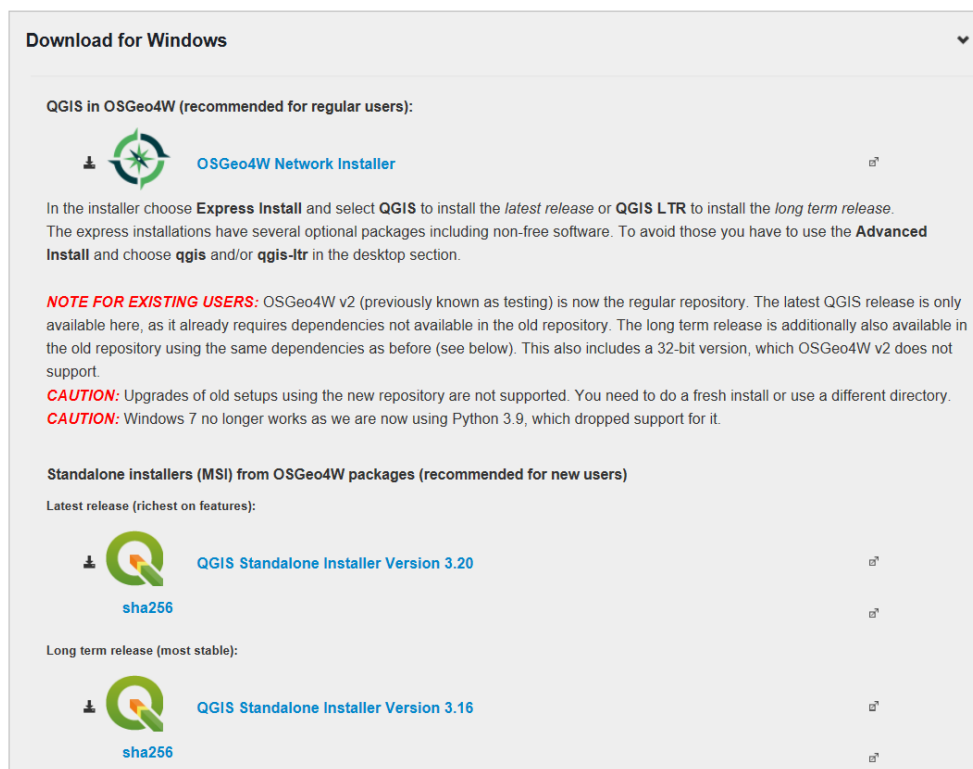


Figure 2.2. QGIS download options.

3. Select '**Save As**' on the Save drop down menu and select a target directory on the computer for the QGIS application to run from. Depending on the operating system you are using, the download may start automatically.
4. Navigate to the folder in which the installation file was saved and double-click on it to begin the installation process. If the download started automatically, the file should be saved in the '**Downloads**' folder. Alternatively, an option may appear to click '**Run**' at the bottom of the web browser. The QGIS installation wizard will then start up and guide you through the installation process. There is an option to download three sample datasets (North Carolina Data Set, South Dakota (Spearfish) and Alaska Data Set). You do not need to download these to run the methods presented in this user guide, however they may be useful as test datasets for exploring the broader functionality of QGIS.

Further useful documentation can be found at: <https://www.qgis.org/en/site/>

2.2 Installing R

1. Go to the R website <https://cran.rstudio.com/>, and select the 'Download R for Windows' link (Figure 2.3).

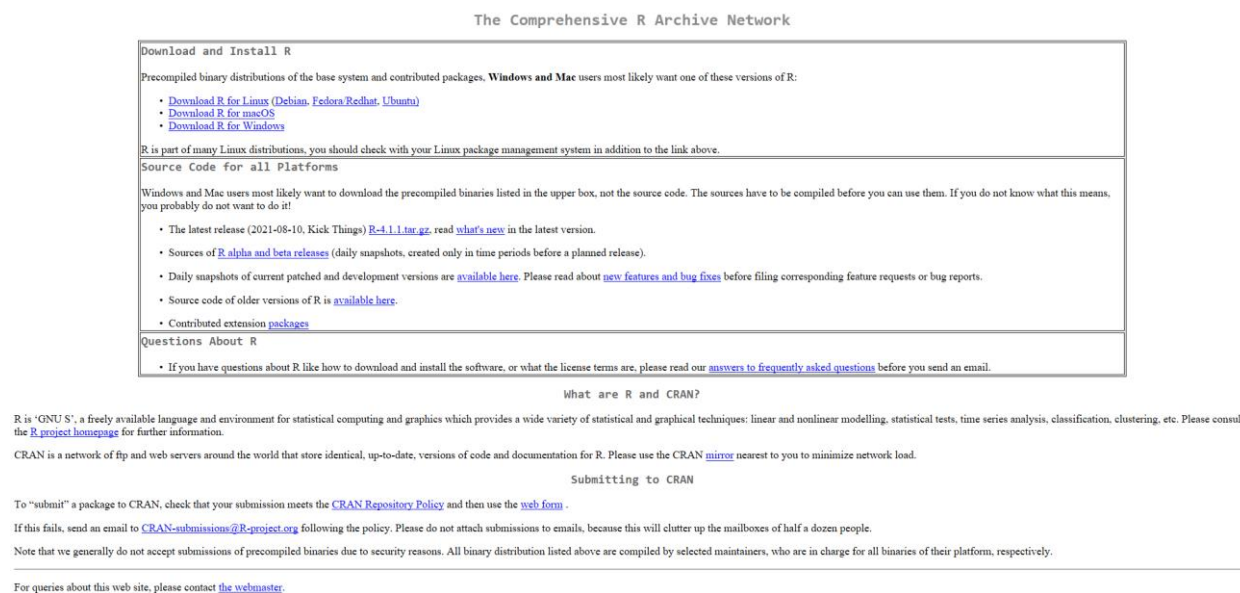


Figure 2.3. Cran.rstudio website with download options.

2. Select the 'base' subdirectory for installing R (Figure 2.4).

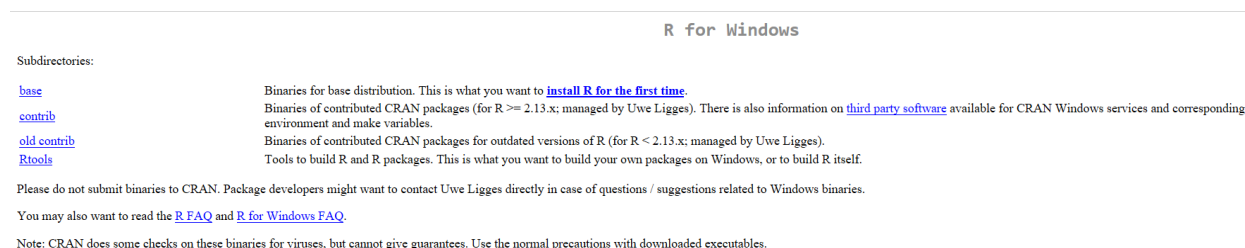


Figure 2.4. R download base subdirectories.

3. Select the 'Download R 4.1.1 for Windows' link (or the most recent version available if version 4.1.1 has been superseded at the time of reading) (Figure 2.5).

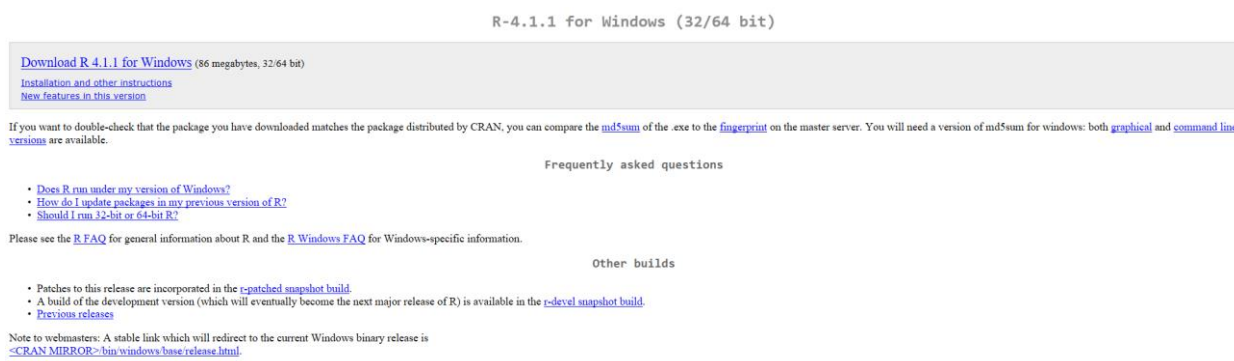


Figure 2.5. R download link.

4. When the installation file has downloaded, double-click on the file to begin the installation process. The installation wizard will then guide you through the installation steps.

2.3 Installing RStudio

RStudio operates as a front-end to R, offering R functionality through a more user-friendly interface. RStudio must be installed separately, after you have installed R.

Installation

1. Go to the '**Download RStudio**' website - <https://www.rstudio.com/products/rstudio/download/>
2. A number of download options are available, but it is the RStudio Desktop (Open Source Licence) that is required here. This option is free to download. Click on the '**Download**' icon under this option (Figure 2.6).



Figure 2.6. RStudio download webpage.

3. On the next window, click the '**Download RStudio for Windows**' icon (Figure 2.7).

RStudio Desktop 1.4.1717 - [Release Notes](#)

1. Install R. RStudio requires R 3.0.1+.
2. Download RStudio Desktop. Recommended for your system:



Requires Windows 10 (64-bit)

Figure 2.7. Download RStudio of windows link.

4. Save the installation file to an appropriate location. Once downloaded, navigate to the location where it is saved to, and double-click on the installation file to begin the installation. Alternatively, depending on the web browser being used, an option may appear to run the installation at the bottom of the web browser page.
5. The RStudio installation wizard will then guide you through the installation steps.

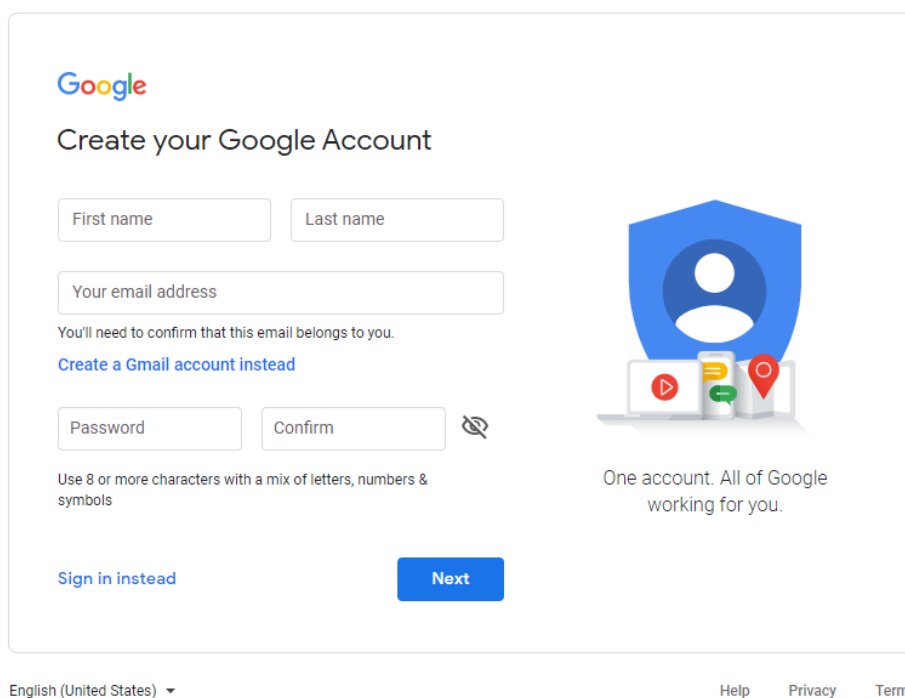
Further useful documentation can be found at: www.rstudio.com and <https://cran.r-project.org/bin/windows/Rtools/>

2.4 Registration for Google Earth Engine

User registration is required to use Google Earth Engine, although this is free for non-commercial applications. To do this, it is necessary to register for a Google Drive account.

2.4.1 Registration for Google Drive

1. Open a web browser and navigate to:
accounts.google.com/SignUpWithoutGmail
2. Enter your name, email address and set a password (Figure 2.8).



The image shows the Google account registration page. At the top left is the Google logo. Below it is the heading "Create your Google Account". There are four input fields: "First name", "Last name", "Your email address", and "Password". The "Password" field has a "Confirm" field next to it and an eye icon to toggle visibility. Below the "Your email address" field, there is a note: "You'll need to confirm that this email belongs to you." and a link: "Create a Gmail account instead". Below the "Password" field, there is a note: "Use 8 or more characters with a mix of letters, numbers & symbols". At the bottom left is a link: "Sign in instead". At the bottom right is a blue "Next" button. On the right side of the form, there is a graphic of a blue shield with a white person icon, and below it, icons for YouTube, Gmail, and Google Maps. Below the graphic is the text: "One account. All of Google working for you." At the bottom of the page, there is a language selector: "English (United States)" and links for "Help", "Privacy", and "Terms".

Figure 2.8. Google account registration webpage.

3. Enter the verification code sent to the email account.

4. Adding a mobile phone number is required at this stage as another method of verification. Upon receiving a text message with the verification code, enter that into the second box.
5. Fill out the personal information box to finish the account creation. Following this, agree to the Terms & Conditions, alongside the Privacy Policy to finish creating an account.

To then access your Google Drive account, enter the following URL into the web browser – <https://www.google.co.uk/drive/> and select the ‘**Go to Google Drive**’ option. Sign in with your newly created login details. You will then have access to your Google Drive and the contents within it. It is here that the datasets that you subsequently create in Google Earth Engine will be exported to.

2.4.2 Registration for Google Earth Engine

1. If you are not logged in already, log in to Google with your Google Account, and enter the following URL into the URL bar <https://earthengine.google.com/>.
2. Click on the ‘**Sign up**’ button in the top right of the page (Figure 2.9).

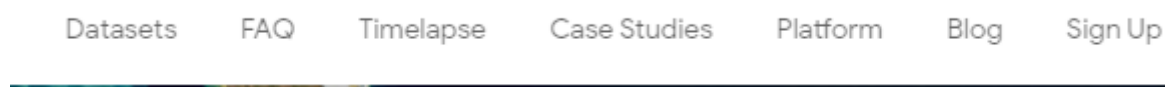


Figure 2.9. Google Earth Engine ‘Sign-up’ button.

3. Complete the form shown in Figure 2.10 with your relevant information.

The image shows a web form titled "Sign up for Earth Engine". At the top, there is a disclaimer: "If you'd like to become an Earth Engine developer, please sign up by providing the following information. We can't accept all applications, so please fill out all fields as best you can so we can evaluate your request for access. If you are accepted, you will receive an email within one week." The form fields include: "Email" (with the value "Anye.bowman1@gmail.com"), "Full name *", "Affiliation/Institution *" (with a sub-note: "Which organization are you a part of? Give a homepage URL, if possible."), "Institution type *" (with a sub-note: "Select the best description for your institution, or choose Other and clarify."), "Country/Region *" (with the value "United States" and a sub-note: "Please tell us where you live."), and "What would you like to accomplish with Earth Engine? *" (with a sub-note: "Please describe in a few sentences how you intend to use Earth Engine."). Below the form, there is a disclaimer: "Earth Engine may only be used for development, research, or educational purposes. It may not be used for sustained commercial purposes, but may be evaluated in a production environment." This is followed by two checkboxes: "I agree that my use of the Earth Engine services and related APIs is subject to my compliance with the applicable Terms of Service. In particular, I acknowledge that creating multiple Earth Engine accounts to circumvent quota restrictions is a violation of the Terms of Service." and "I am interested in commercial use of Earth Engine." At the bottom, there is a checkbox for "I'm not a robot" and a reCAPTCHA logo with links for "Privacy" and "Terms".

Figure 2.10. Google Earth Engine sign-up form.

4. A registration confirmation email will be sent to the email account used to register the account. Open the registration email and click on the '**Earth Engine Code Editor**' link contained in it to access the code editor. This registration email may take some time to come through, so we would encourage users to allow ample time between submitting the registration and the time at which you need to conduct the analysis.

The registration email (Figure 2.11) contains links to the Earth Engine Code Editor in which the analysis will be performed, but also the Earth Engine API, which contains a wealth of reference information about GEE functionality and datasets. It also contains a variety of other links including frequently asked questions and additional tutorials and documentation. It is worthwhile users taking some time to explore these resources for a broader introduction to the capabilities of GEE.

Welcome to Earth Engine!



Greetings, Earth Engine Developer, and welcome! You will soon have access to:

- The [Earth Engine Code Editor](#) - the primary Earth Engine development environment.
- The [Earth Engine API](#) - including our [Python library](#).
- The [Earth Engine Explorer](#) - a graphical user interface. No programming skills needed.

Note that it may take a few days before this change is propagated through the system.

To get started with Earth Engine, we suggest you:

- Read our [Frequently Asked Questions](#).
- Check out our [Get Started](#) guide, [tutorials](#), and complete [documentation](#).
- Visit the Earth Engine [developers list](#).

It's great to have you on board. We look forward to seeing what you can do with Earth Engine!

Figure 2.11. GEE registration email with relevant links.

3. Study area

The example presented in this user guide focuses on a study area of the medium-sized town of **Lodja** (population c. 80,000) (latitude: -3.524661°, longitude: 23.596669°) and surrounds in the **Democratic Republic of the Congo**, which is meso-endemic for malaria (Figure 3.1). The area surrounding Lodja is typified by a mix of land cover components including traditional small holder shifting cultivation (cleared land, active field, fallow fields) along with settlements, grassy and bare areas, and a permeable interface area with forest. The Lukene river flows immediately to the south of Lodja. Mean monthly rainfall (between 1991-2015) was <100 mm for June and July and 100-220 mm/month for the other months of the year, with mean temperatures of 24-26 °C all year round.

Mosquito survey data provided in the .csv file details *Anopheles gambiae* abundance collected monthly from 8 houses over 2015 and 2016.

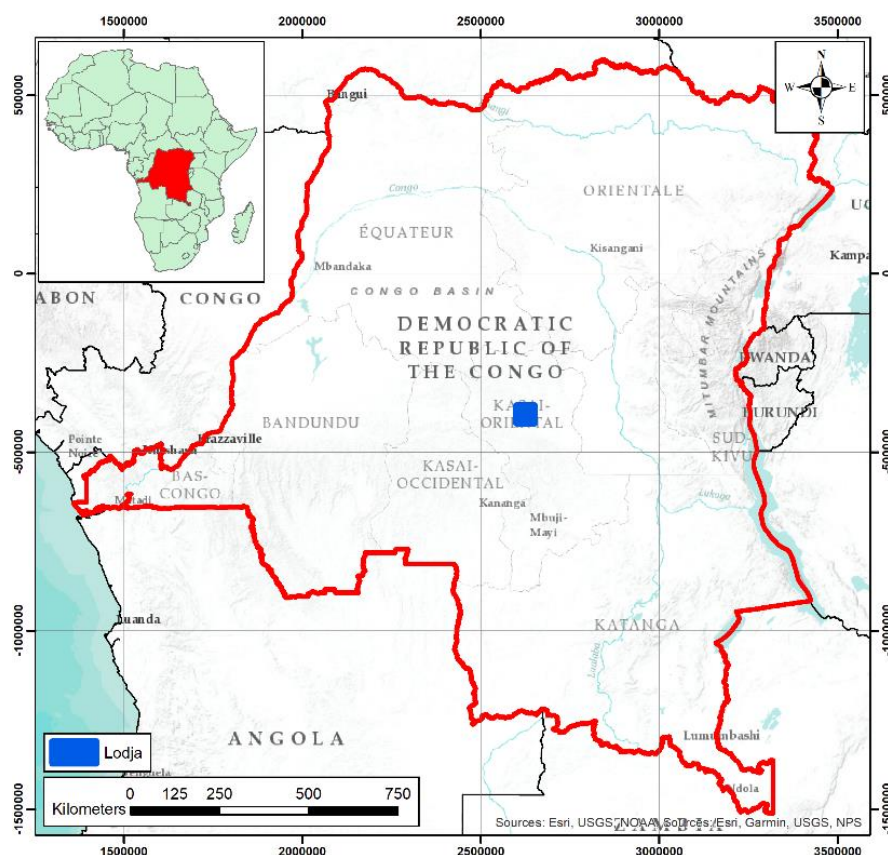


Figure 3.1 Lodja study area location.

4. Dataset generation – QGIS

Most of the workflow in this user manual draws on the ingested data archives of GEE, but other input data, specific to the study area of interest, are also required. This includes:

- 1) Coordinates, or a shapefile delineating the study area of interest.
- 2) A polygon shapefile corresponding to training areas for generating a land cover classification of the study area. Each training polygon requires an integer code/attribute corresponding to the appropriate land cover class for that polygon (e.g.1 forest, 2 grassland etc. see table 4.1).
- 3) A validation data set of locations of known land cover types to perform accuracy assessment of the land cover classification. This shapefile should have points containing integer code/attributes that correspond to the land cover class at that location. The integer codes used for each land cover class must be consistent between the training (polygon) and validation (point) shapefiles.
- 4) Mosquito survey data including survey location coordinates, survey date and mosquito species and abundance for each location.


It is expected that the mosquito count data will be available as a .csv file (viewable and editable in Excel), with columns corresponding to different data attributes, and rows corresponding to individual survey records (Figure 4.1). Data survey locations recorded with X and Y coordinates as different attributes are a pre-requisite for analysis.

	A	B	C	D	E	F
1	Long_dd	Lat_dd	Year	Month	House	An_gambiae_total
2	23.5830333	-3.5394833	2016	January	1	6
3	23.5830000	-3.5397167	2016	January	2	21
4	23.5836833	-3.5402833	2016	January	3	84
5	23.5837000	-3.5395333	2016	January	4	87
6	23.5838167	-3.5394000	2016	January	5	33
7	23.5839333	-3.5393000	2016	January	6	4
8	23.5833500	-3.5389167	2016	January	7	5
9	23.5999500	-3.5394167	2016	January	8	24
10	23.5830333	-3.5394833	2016	February	1	47
11	23.5830000	-3.5397167	2016	February	2	67
12	23.5836833	-3.5402833	2016	February	3	94
13	23.5837000	-3.5395333	2016	February	4	14
14	23.5838167	-3.5394000	2016	February	5	7
15	23.5839333	-3.5393000	2016	February	6	17
16	23.5833500	-3.5389167	2016	February	7	7
17	23.5999500	-3.5394167	2016	February	8	5

Figure 4.1. Example of mosquito survey data format.

4.1 Creating a validation point shapefile

Here, we will create a point shapefile for validation comprising locations of known land cover types to assess the accuracy of the land cover classification that we will produce. Point shapefiles can be created manually (see the section below where a shapefile is created and populated; note that for the validation data, a point rather than polygon shapefile would be used) or generated from pre-existing ground reference points typically in spreadsheet format. We will create a point shapefile of validation data from a pre-existing dataset saved in .csv format. To import from a spreadsheet, follow these steps:

- Open **QGIS** and create a new project by clicking on the  icon under the main toolbar.
- On the main **'Layer'** toolbar, select **'Add Layer'**, then **'Add Delimited Text Layer'** (Figure 4.2). Select the .csv file to import in the 'File name' box.
- Click on the **'Geometry Definition'** option. Make sure 'Point coordinates' is selected, then for the **'X field'** select **'Long_dd'**, and for **'Y field'** select **'Lat_dd'**.
- Ensure X and Y co-ordinates are in the correct columns and keep the default parameters.

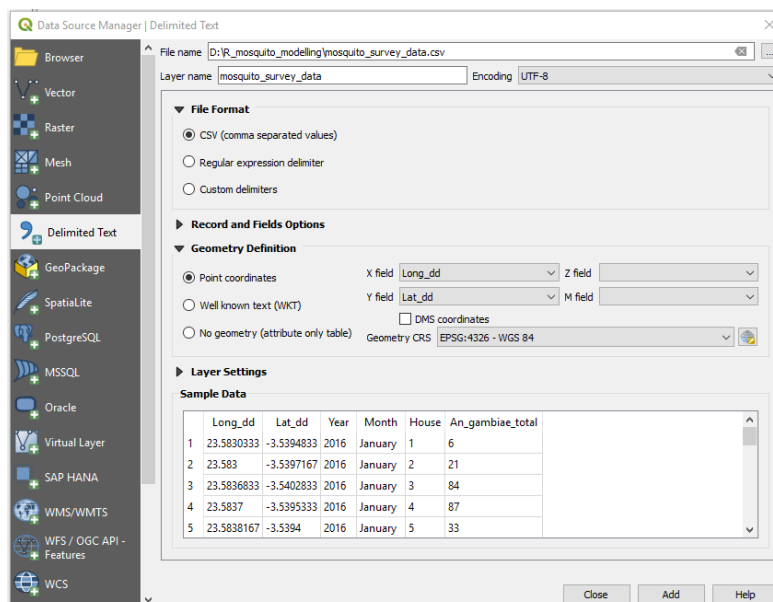



Figure 4.2. QGIS 'Create a layer from a delimited text file' window.

- Next, we check that the appropriate coordinate reference system (CRS) is being used. As our coordinates here are in decimal degrees, we use the CRS EPSG:4326 – WGS84. Check that this is selected as the **‘Geometry CRS’**. If it is not, set the appropriate coordinate reference system by clicking the  icon next to the **‘Geometry CRS’** drop-down box. This will open the **‘Coordinate Reference System Selector’** window (Figure 4.3).

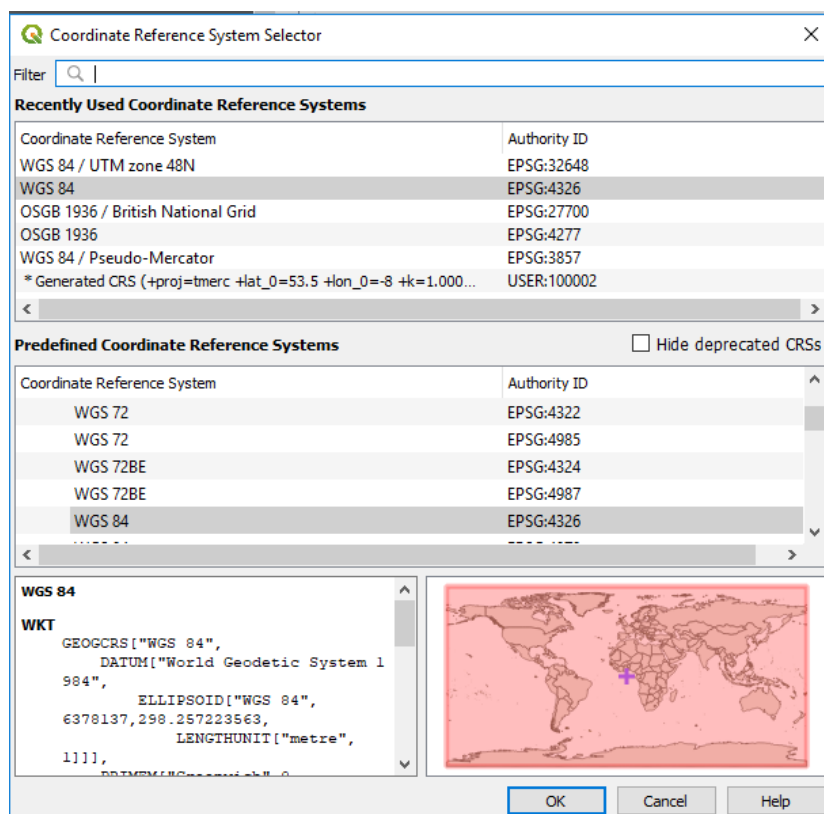


Figure 4.3. The coordinate reference system selector window.

- Here, we can use the **‘Filter’** search tab at the top of the window to find the required coordinate system. Type 4326 into the search filter, and the WGS84 EPSG:4326 option will appear below the filter bar. Select this and click **‘OK’**.
- Returning to the **‘Data Source Manager’** window, click **‘Add’** then close, and the locations in the .csv file should then be displayed (Figure 4.4). The layer name **‘mosquito_survey_data’** should be displayed in the **‘Layers’** panel.

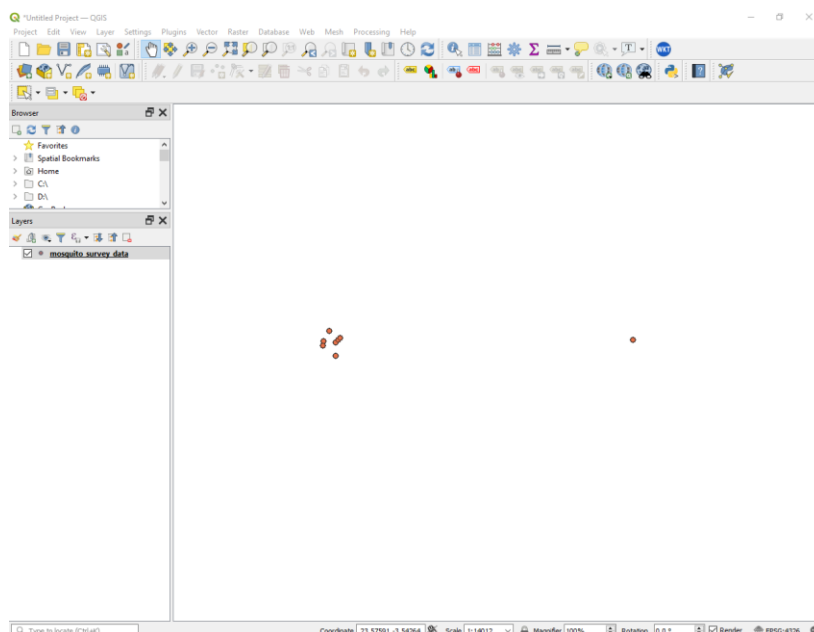



Figure 4.4. The locations contained in the .csv file displayed in QGIS.

- To then save the mosquito_survey_data layer as a shapefile, right-click on the layer in the 'Layers' panel, select 'Export' then 'Save Features As...'. Specify the File Format as 'ESRI Shapefile' and then specify the file name of the shapefile to be saved (you can retain the mosquito_survey_data file name), and specify the location where the file is to be saved using the  icon.

4.2 Creating a training polygon shapefile

One of the steps in this tutorial is to generate a land cover classification of the study area. Depending on the specific nature of the study area, the land cover classes appropriate to map may differ. In this example, an eight-class classification nomenclature will be used with these classes and the corresponding land cover codes displayed in Table 4.1. This user guide **comes complete with training and validation data for the classification for the study example Lodja** but if users wish to create their own for their area of interest sections 4.2 and 4.3 describe the steps on how to do this.

Table 4.1. Land cover classes and corresponding codes.

Land cover class	Code
Forest	1
Grassland	2
Clearing	3
Fallow	4
Built-up	5
Flowing water	6
Static water	7
Burnt	8

Ideally, land cover classification training and validation data will be based on field-collected data where locations of known land cover type are recorded with a GPS. This can, where appropriate, be supplemented with additional training / validation data derived from very-high resolution (VHR) imagery of the area of interest derived via public portals such as Google Earth or Bing Aerial. Users must be aware of the limitations of using these data sources especially where there is a temporal mismatch between the acquisition dates of the satellite data that is being classified, and the VHR imagery being used as the reference. We do not offer a review of classification or training/validation data collection methodology here, but we would encourage users to engage with appropriate background literature on these topics to improve their understanding of the process, and how differing approaches can impact the quality of the output land cover classifications.

The training dataset for the land cover classification that will be performed in **Google Earth Engine** will be a point vector dataset containing an equal number of training points (locations) for each land cover class. Initially, a polygon shapefile will be created with multiple polygons corresponding to areas of known land cover types. These training polygons will be created for each land cover class that we wish to classify. As the distribution of land cover class across a study area varies with some land cover types scarcer than others, and as the size and number of polygons created for each land cover class is likely to vary, we then create a subset of training data from these polygons. This process takes the existing training polygons and selects a defined number of randomly located points within the spatial extent of the polygons for each land cover class. This ensures that a balanced training dataset is used for the

classification, with each land cover class equally represented in the training data. The training polygon dataset will be imported into GEE as an asset, and the sub-setting and conversion to a point dataset will be performed within GEE.

Here, reference VHR imagery layers available in QGIS will be used as a reference data source for creating the initial polygon training shapefile.

- Open **QGIS** and install the '**QuickMapServices**' plugin, which contains a variety of data sources including VHR imagery that can be used as a basemap for contextualisation, and for reference data collection. To do this, click '**Plugins**' on the main toolbar, then '**Manage and install plugins**'.
- The '**Plugins**' pop-up box should open (Figure 4.5). In the search bar, search for '**QuickMapServices**', then install the plugin.

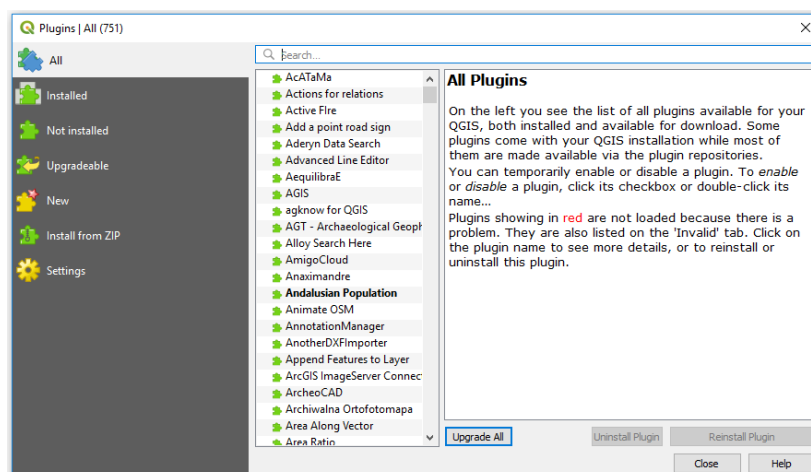


Figure 4.5. QGIS Plugins window.

- Once installed, the QuickMapServices option should appear in the '**Web**' tab on the main toolbar. If you select this, it will display the layers that are currently available to display as base maps (Figure 4.6). This currently only gives a limited number of options – we can increase these options by downloading additional contributed packs.

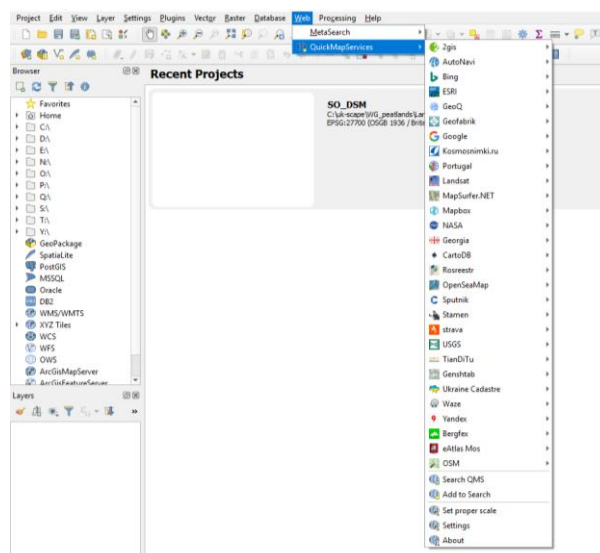



Figure 4.6 QGIS QuickMapServices layer options.

- To do this, open the **'QuickMapServices'** option from the main toolbar, and click the **'settings'** option. On the pop-up box that opens, select the **'More services'** tab, then click on **'Get contributed pack'**. Click **'Save'**. There should now be a larger range of base maps and imagery products available via QuickMapServices, including VHR imagery from the Google Satellite and Bing satellite layers. Select **'Google Satellite'** and this should appear as a basemap, and as an option in the **'Layers panel'**, which can be switched on or off. If no other data is loaded in the QGIS session then the basemap will initially display for the full globe, however you can use the pan and zoom functions to focus on your area of interest at a much higher level of detail.
- Next, create a **new project** by clicking on the icon under the main toolbar. Then, click on the **'New Shapefile Layer'**  icon under the main toolbar, and the **'New Shapefile Layer'** window should open (Figure 4.7).

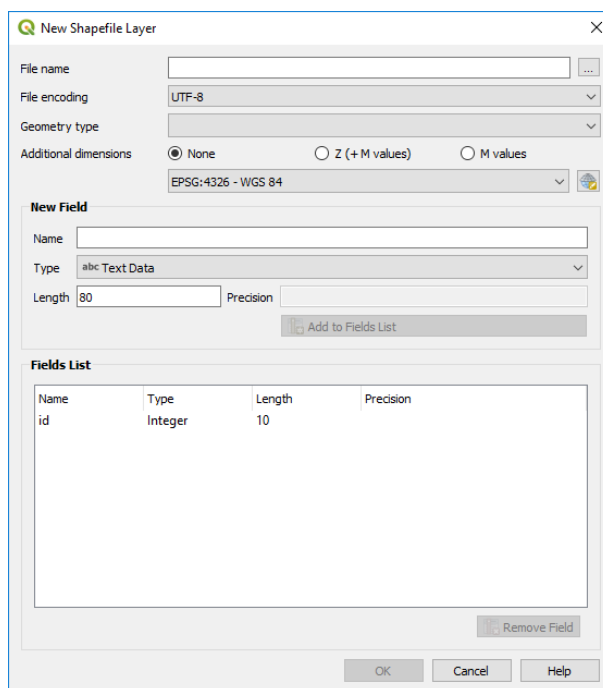




Figure 4.7. New shapefile window.

- Fill out the menu box, ensuring you set the geometry '**Type**' to '**Polygon**' and select the appropriate Coordinate Reference System. You then need to add a column for the land cover classes in the '**New field**' section. In the '**Name**' box type '**Class**'. Make sure the '**Type**' drop down menu is set to '**Whole number**'. Press '**Add to fields list**' to add this to the polygon.
- Save the polygon into an appropriate directory. It should then be added as a layer in the '**Layers**' tab.
- Select the shapefile in the Table of Contents, and then click the '**Toggle Editing**'  button on the toolbar on the top panel. Then select the '**Add Feature**'  button underneath.
- Draw around an area of interest for a particular land cover class, using the left mouse button to add vertices and a right click to complete the polygon. When the polygon is finished, the '**Feature Attributes**' box required details to be added (Figure 4.8). Add the polygon id and the class number that corresponds to the land cover class (Table 4.1) being digitized.

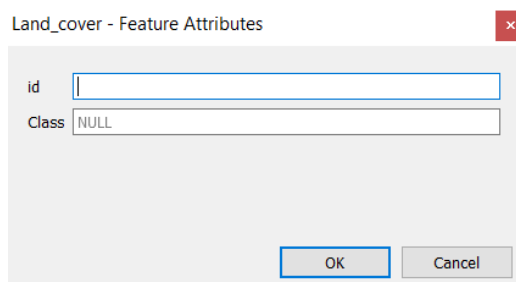




Figure 4.8. Feature Attributes window.

- Continue adding more training polygons until a full training dataset is complete.
- To finish editing, save the polygons by hitting the **'Save'**  button and then finish editing by clicking the **'Toggle Editing'**  button. It is recommended that you save your edits at regular intervals throughout this process.

4.3 To create the study area extent shapefile

We also need to create a shapefile delineating the study area of interest to constrain the area of data processed in, and exported from, Google Earth Engine. To do this, follow the steps set out in the **'4.2 Creating a training polygon shapefile'** section to create a new shapefile. Then add a single polygon corresponding to the area of interest and save the polygon. It is not necessary to add further attributes to this shapefile, although you can do if you so wish.

5. Google Earth Engine - satellite data pre-processing

5.1 GEE interface / GUI

Google Earth Engine (GEE) is the cloud-processing platform through which much of the data pre-processing is performed prior to the subsequent modelling stage. Before getting started with the analysis, lets familiarise ourselves with the GEE interface.

First, open a web-browsing session in Google Chrome and navigate to <https://code.earthengine.google.com>. The GEE interface appears as in Figure 5.1, here with an example script and analysis output displayed.

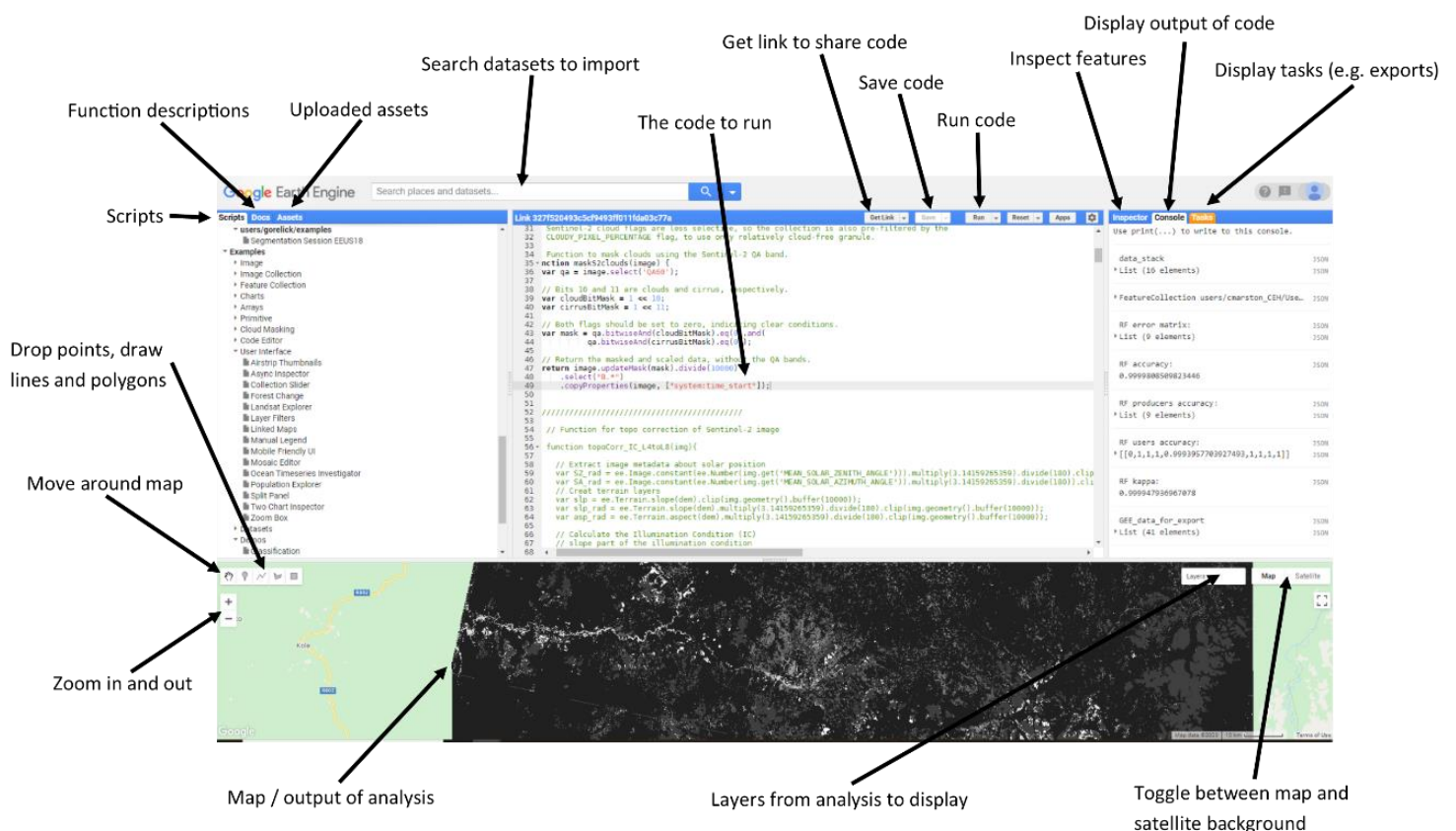


Figure 5.1. Google Earth Engine interface.

GEE works via command-line using the programming language Javascript, with specific commands given as lines of code corresponding to different data processing

UKCEH report ... version 1.0

applications. Multiple commands can be linked together for more complex multi-stage data processing tasks, with scripts sharable with other users. It is advised to save revisions made to scripts at regular intervals. If you would like to share scripts that you have developed with other users you can do this by clicking '**Get Link**', then in the pop-up window that opens click the '**Click to copy link**' icon. You can then distribute the link as appropriate to other users who will be able to access the script. If the script relies on assets that have been uploaded to a user account, it will be necessary to share those assets. To do this, navigate to the appropriate location in the '**Assets**' tab, hover the cursor over the appropriate asset filename and click on the '**share**' icon. In the pop-up box that opens tick the '**Anyone can read**' check box, then click '**Done**'. The asset will then be available for other users to access when they are running the script that you have distributed.

To open a GEE script that has been shared with you, simply paste the link into a Chrome web browser session or double-click the hyperlink. You will then be taken to a GEE code editor session displaying the script which you will then be able to run.

Although this tutorial uses a script developed to run analysis similar to that contained in Marston *et al.* 2023, a far broader range of functionality and datasets are available within GEE and users are encouraged to investigate these via the GEE website at <https://earthengine.google.com/>.

5.2 Uploading assets

Some of the datasets that will be modelled in response to mosquito abundance can be directly exported from GEE, however others (such as land cover proportional coverage and distance to nearest patch of a given land cover type) will be generated by the user within GEE. To do this, it is necessary to upload additional datasets into GEE to perform and validate the land cover classifications. These datasets must be uploaded as assets and will include shapefiles delineation of the area of interest (AOI), training areas of known land cover type for performing land cover classification, and validation locations for assessing the accuracy of the land cover classifications performed. Points and/or polygons can be added directly via the GEE, however, as datasets such as these are often collected in advance during fieldwork campaigns and are often large in number, a more practical way to add this data into GEE is to load an existing shapefile

containing the required data as an asset. To upload an asset, in the GEE session select the '**Assets**' tab (top left), click '**New**' then '**Shapefiles**'. Click '**Select**', navigate to the location where the shapefile is saved. Although shapefiles are viewed as a single file in a GIS, they are actually comprised of several files, which will have the same prefix (filename), but different file extensions, for example **.shp**, **.shx**, **.dbf** and others. These multiple files work in combination to define the geometry and attributes of the data to be displayed or analysed. When uploading a shapefile to GEE, all the constituent files forming the shapefile must be selected, with the exception of the **.sbx file**. If the **.sbx** file is selected, then the upload process will fail. If this occurs, try uploading the shapefile again without including the individual file with the file extension that causes the problem.

On this occasion, we will require training data identifying locations of known land cover types for performing a land cover classification, and a separate validation dataset to perform an accuracy assessment of the classification. These have already been uploaded as two separate assets and shared, the training data as a polygon shapefile, and the validation data as a point shapefile.

5.3 GEE scripts

Three Google Earth Engine scripts have been provided accompanying this user manual and are used for the following sequential stages;

1. Data preparation (page 26)
2. Data modelling (page 63)
3. Data visualisation (page 71)

The scripts are made available via the hyperlinks provided in the relevant sections of this user manual. If you use **ctrl and click**, this link it will open a GEE code editor session containing the script. Alternatively, this link can be copied and pasted into a web browser session.

Script 1: Data pre-processing

<https://code.earthengine.google.com/d1e89a3d439d14e84df704a7c81d442c>

The code will appear similar to that displayed in Figure 5.2.

Earth observation for malaria modelling: a practical toolkit for satellite-based prediction of mosquito distributions using Google Earth Engine and R

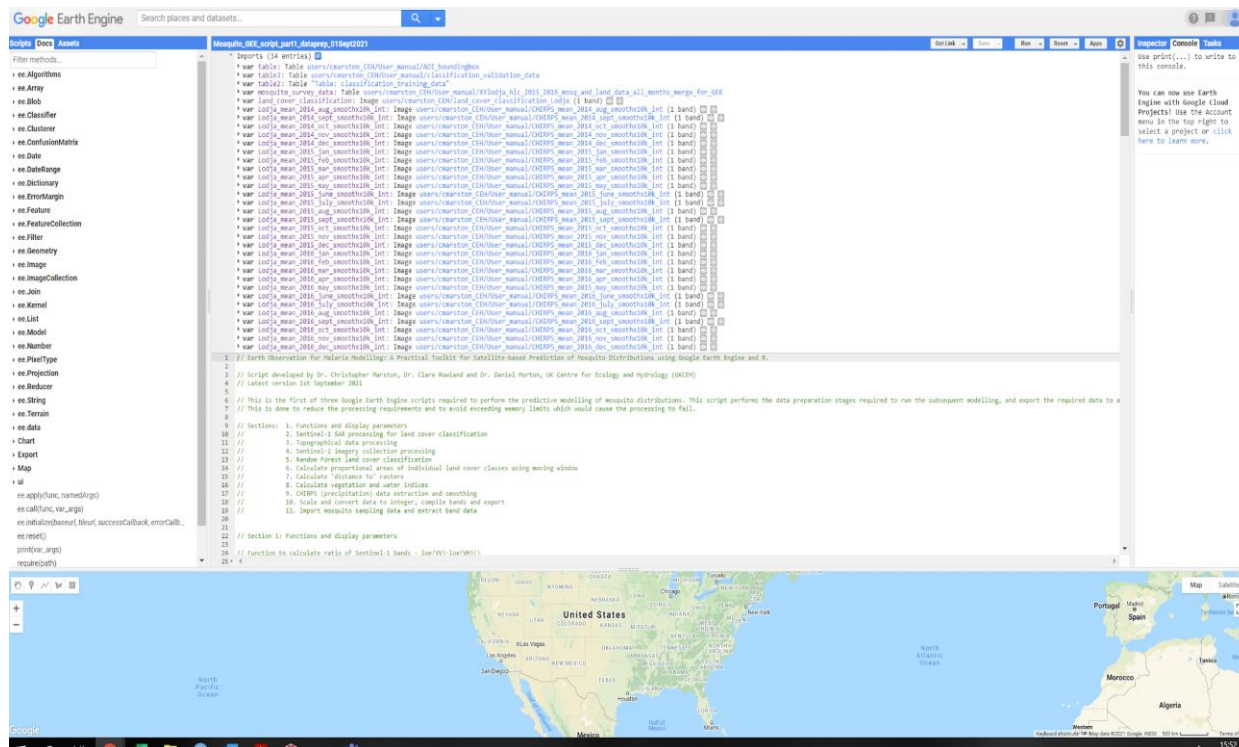


Figure 5.2. Google Earth Engine code editor with script displayed.

The following sections provide an explanation for the various stages of the data processing and modelling. Instructions on running the scripts is found in section 5.5 (p.52)

The GEE data processing requires several steps, which are outlined below. The code required to run these steps is described below but note that where an example is given (for example to perform a stage of data analysis for a particular land cover class), this may need repeating for other variables of interest (e.g., other land cover classes). For brevity, a single example is provided and described in this document, however the accompanying GEE script contains the complete code. The GEE script is divided into sections as following:

- 2 Functions and display parameters
- 3 Sentinel-1 SAR processing for land cover classification
- 4 Topographical data processing
- 5 Sentinel-2 imagery collection processing
- 6 Random Forest land cover classification

- 7 Calculate proportional areas of individual land cover classes using moving window
- 8 Calculate 'distance to' rasters
- 9 Calculate vegetation and water indices
- 10 CHIRPS (precipitation) data extraction and smoothing
- 11 Scale and convert data to integer, compile bands and export
- 12 Import mosquito sampling data and extract band data

Sections of the script denoted by `//` are 'commented out', these are recognised by GEE as being inert code i.e. descriptive text rather than a command to run (e.g. Figure 5.3). This is useful for adding in comments or notes to describe what the commands do at certain stages of the script, or for disabling commands without deleting them from the script where this would be useful. Sections of the script that are commented out are displayed in green text.

```
22 // Section 1: Functions and display parameters
23
24 // Function to calculate ratio of Sentinel-1 bands - log(VV)-log(VH)()
25 var vh_vv = function(image) {
26   return image.select('VH').divide(image.select('VV'));
27 };
28
```

Figure 5.3. Example of the script with `//` and green text denoting comments

Although the content of this user-manual and the code therein are based on the research contained in Marston *et al.* (2023), there are slight differences in the methods and implementation of that paper and this user guide. These adaptations are deliberate and designed to utilise the richer abundance of satellite Earth Observation dataset available since the time period of field data collection involved in Marston *et al.* (2023), with a view to the future implementation of these methods across other malaria endemic regions. Principally, whereas Marston *et al.* (2023) utilised single cloud free images, the methods presented in this user guide instead use collections of images in combination to generate cloud-free composites, offering improved flexibility and

opportunity for practical application of these methods in areas where opportunities to obtain single cloud-free images are low.

5.4 Importing assets

Section 5.2 described how you can upload assets for analysis in GEE should you need to do this for your own study sites and applications. To view the assets have been uploaded and are available, click on the **'Assets'** tab towards the top-left of the GEE window (Figure 5.4). This will display the assets that are available for analysis.

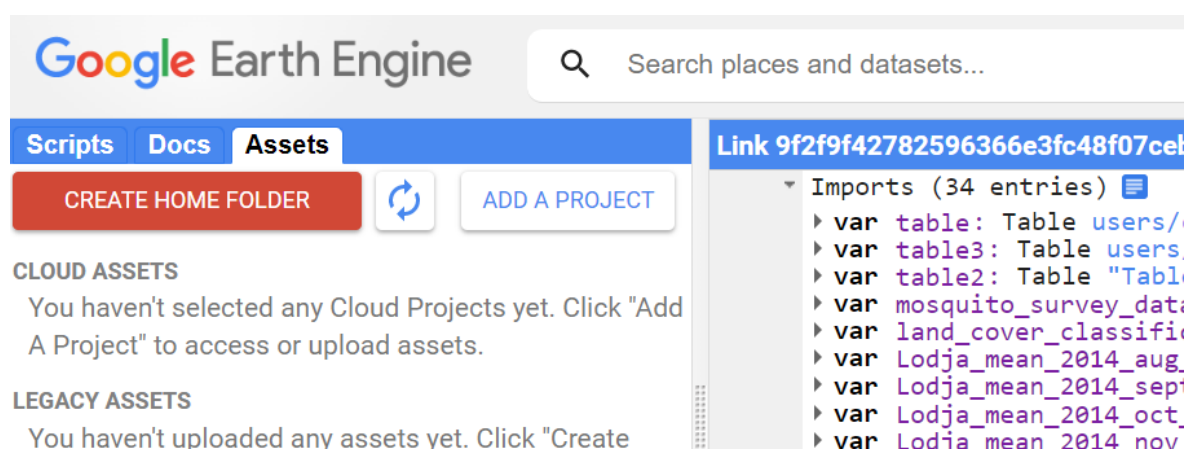


Figure 5.4. Assets tab on the GEE console

If your assets are saved in sub-directories, you can expand the sub-directory to view its contents. When the cursor is hovered over the asset of interest, three icons will appear, these being **'Share'**, **'Rename'** and **'Import into script'**. To make an asset available for analysis using the GEE script, you must import it by clicking on the **'Import into script'** icon. When imported, the asset will then appear at the top of the code window and will be available as an asset that can be called by the code. The name of the asset is also given, for example in Figure 5.5 three assets are imported and have been assigned the names table, table2 and table3.

```
Imports (34 entries)
var table: Table users/cmarston_CEH/User_manual/AOI_boundingbox
var table3: Table users/cmarston_CEH/User_manual/classification_validation_data
var table2: Table "Table: classification_training_data"
var mosquito_survey_data: Table users/cmarston_CEH/User_manual/XVlodja_hlc_2015_2016_mosq_and_land_data_all_months_merge_for_GEE
var land_cover_classification: Image users/cmarston_CEH/land_cover_classification_Lodja (1 band)
var Lodja_mean_2014_aug_smoothx10k_int: Image users/cmarston_CEH/User_manual/CHIRPS_mean_2014_aug_smoothx10k_int (1 band)
var Lodja_mean_2014_sept_smoothx10k_int: Image users/cmarston_CEH/User_manual/CHIRPS_mean_2014_sept_smoothx10k_int (1 band)
var Lodja_mean_2014_oct_smoothx10k_int: Image users/cmarston_CEH/User_manual/CHIRPS_mean_2014_oct_smoothx10k_int (1 band)
var Lodja_mean_2014_nov_smoothx10k_int: Image users/cmarston_CEH/User_manual/CHIRPS_mean_2014_nov_smoothx10k_int (1 band)
var Lodja_mean_2014_dec_smoothx10k_int: Image users/cmarston_CEH/User_manual/CHIRPS_mean_2014_dec_smoothx10k_int (1 band)
var Lodja_mean_2015_jan_smoothx10k_int: Image users/cmarston_CEH/User_manual/CHIRPS_mean_2015_jan_smoothx10k_int (1 band)
var Lodja_mean_2015_feb_smoothx10k_int: Image users/cmarston_CEH/User_manual/CHIRPS_mean_2015_feb_smoothx10k_int (1 band)
var Lodja_mean_2015_mar_smoothx10k_int: Image users/cmarston_CEH/User_manual/CHIRPS_mean_2015_mar_smoothx10k_int (1 band)
var Lodja_mean_2015_apr_smoothx10k_int: Image users/cmarston_CEH/User_manual/CHIRPS_mean_2015_apr_smoothx10k_int (1 band)
var Lodja_mean_2015_may_smoothx10k_int: Image users/cmarston_CEH/User_manual/CHIRPS_mean_2015_may_smoothx10k_int (1 band)
var Lodja_mean_2015_june_smoothx10k_int: Image users/cmarston_CEH/User_manual/CHIRPS_mean_2015_june_smoothx10k_int (1 band)
var Lodja_mean_2015_july_smoothx10k_int: Image users/cmarston_CEH/User_manual/CHIRPS_mean_2015_july_smoothx10k_int (1 band)
var Lodja_mean_2015_aug_smoothx10k_int: Image users/cmarston_CEH/User_manual/CHIRPS_mean_2015_aug_smoothx10k_int (1 band)
var Lodja_mean_2015_sept_smoothx10k_int: Image users/cmarston_CEH/User_manual/CHIRPS_mean_2015_sept_smoothx10k_int (1 band)
var Lodja_mean_2015_oct_smoothx10k_int: Image users/cmarston_CEH/User_manual/CHIRPS_mean_2015_oct_smoothx10k_int (1 band)
var Lodja_mean_2015_nov_smoothx10k_int: Image users/cmarston_CEH/User_manual/CHIRPS_mean_2015_nov_smoothx10k_int (1 band)
var Lodja_mean_2015_dec_smoothx10k_int: Image users/cmarston_CEH/User_manual/CHIRPS_mean_2015_dec_smoothx10k_int (1 band)
var Lodja_mean_2016_jan_smoothx10k_int: Image users/cmarston_CEH/User_manual/CHIRPS_mean_2016_jan_smoothx10k_int (1 band)
var Lodja_mean_2016_feb_smoothx10k_int: Image users/cmarston_CEH/User_manual/CHIRPS_mean_2016_feb_smoothx10k_int (1 band)
var Lodja_mean_2016_mar_smoothx10k_int: Image users/cmarston_CEH/User_manual/CHIRPS_mean_2016_mar_smoothx10k_int (1 band)
var Lodja_mean_2016_apr_smoothx10k_int: Image users/cmarston_CEH/User_manual/CHIRPS_mean_2016_apr_smoothx10k_int (1 band)
var Lodja_mean_2016_may_smoothx10k_int: Image users/cmarston_CEH/User_manual/CHIRPS_mean_2016_may_smoothx10k_int (1 band)
var Lodja_mean_2016_june_smoothx10k_int: Image users/cmarston_CEH/User_manual/CHIRPS_mean_2016_june_smoothx10k_int (1 band)
var Lodja_mean_2016_july_smoothx10k_int: Image users/cmarston_CEH/User_manual/CHIRPS_mean_2016_july_smoothx10k_int (1 band)
var Lodja_mean_2016_aug_smoothx10k_int: Image users/cmarston_CEH/User_manual/CHIRPS_mean_2016_aug_smoothx10k_int (1 band)
var Lodja_mean_2016_sept_smoothx10k_int: Image users/cmarston_CEH/User_manual/CHIRPS_mean_2016_sept_smoothx10k_int (1 band)
var Lodja_mean_2016_oct_smoothx10k_int: Image users/cmarston_CEH/User_manual/CHIRPS_mean_2016_oct_smoothx10k_int (1 band)
var Lodja_mean_2016_nov_smoothx10k_int: Image users/cmarston_CEH/User_manual/CHIRPS_mean_2016_nov_smoothx10k_int (1 band)
var Lodja_mean_2016_dec_smoothx10k_int: Image users/cmarston_CEH/User_manual/CHIRPS_mean_2016_dec_smoothx10k_int (1 band)
```

Figure 5.5. Assets imported into the GEE script.

For the purposes of this training exercise, the required assets have already been uploaded and are read in at the beginning of the relevant GEE script.

5.3 Description of the script

5.3.1 Functions and display parameters

To run a particular command on every feature within a collection, we first define the operation that we wish to apply to every element in the collection as a function. We are then able to apply, or ‘map’ that specified function over every object or image in a specified collection using the `map()` command. For example, if the function is to generate a **Normalised Difference Vegetation Index (NDVI)** layer for a particular image, and the collection contains all Sentinel-2 images for a location over a year, then mapping the function over the collection will generate NDVI layers for each image in the collection.

In this example, the functions used within the script do not require editing. As such, they are not examined in detail here other than to list what the functions do:

- Calculate a ratio (VV/VH) layer from Sentinel-1 VV and VH polarisation bands;
- Cloud-mask the Sentinel-2 imagery and mask image edges

- Calculate Normalised Difference Vegetation Index (NDVI)
- Calculate Soil-adjusted Vegetation Index (SAVI)
- Calculate Normalised Difference Water Index (NDWI)
- Calculate Modified Normalised Difference Water Index (MNDWI)

These functions will be mapped across data collections at appropriate stages throughout the script.

Next, we pre-define a colour palette that can be used to display the land cover classification that we will generate for inspection. A specific colour is defined for each class that will be mapped using the code below. The command `var` instructs GEE to create a new object, and this followed by the name of the object to be created which here we will call `palette`. The following code then specifies what the object created will be – here it will comprise a series of hex codes each of which correspond to a colour. As there will be eight land cover classes in the classification, eight hex codes are specified. Each code is then followed by commented out text identifying which class the specific colour applies to, what that class is, and the colour that will be displayed. As this text is commented out these are not commands that GEE will run, but they do provide a useful reference note to the user. More details on hex codes and the corresponding colours can be found at <https://cloford.com/resources/colours/500col.htm>

```
var palette = [  
  '008000', // Class 1, forest, green  
  'FFFF00', // Class 2, grassland, yellow  
  'C0C0C0', // Class 3, clearing, silver  
  '800000', // Class 4, fallow, maroon  
  '000000', // Class 5, built-up, black  
  '00FFFF', // Class 6, flowing water, cyan  
  '0000FF', // Class 7, static water, blue  
  '800080', // Class 8, burnt, purple  
];
```

Next, we specify the established flight distance of the mosquito species that we are modelling here, *Anopheles gambiae*. This is specified as 846 m in Verdonschot and Besse-Lototskaya (2014).

```
var flight_distance = 846;
```

Having defined the functions, display parameters and flight distance that are required within the GEE script, we move on to the data preparation stages.

5.3.2 Sentinel-1 SAR processing for land cover classification

The next set of commands selects a collection of **Sentinel-1 Synthetic Aperture Radar (SAR) images**, extracts the VV and VH polarisation data from this collection, calculates a ratio data product, and generates median pixel value data sets for each of the VV, VH and ratio data products. First, we create the collection of Sentinel-1 images.

```
// Get the Sentinel-1 collection.  
  
var collectionVV = ee.ImageCollection('COPERNICUS/S1_GRD')  
  
.filter(ee.Filter.eq('instrumentMode', 'IW'))  
  
.filter(ee.Filter.listContains('transmitterReceiverPolarisation', 'VV'))  
  
.filter(ee.Filter.listContains('transmitterReceiverPolarisation', 'VH'))  
  
.filter(ee.Filter.eq('orbitProperties_pass', 'DESCENDING'))  
  
.filterBounds(table)  
  
.filterDate('2016-10-08', '2017-10-08')
```

Let us break what these commands do down line by line. The first commented out line simply identifies to the user what actions the following block of commands performs.

```
// Get the Sentinel-1 collection.
```

The next line instructs GEE to create a collection of Sentinel-1 SAR images. `var` denotes that a new object is being created by the following command(s). `collectionS1` is the name of the object (the collection of images) that is being created. `=` denotes that the collection is being created based on the following command(s).

`ee.ImageCollection('COPERNICUS/S1_GRD')` denotes that the image collection that we are creating will draw on the GEE ingested archives of Sentinel-1 SAR imagery which has the GEE dataset identifier of 'COPERNICUS/S1_GRD'. A wide range of pre-ingested datasets are available within GEE each with its own identifier. If we wished to perform this function on a different dataset, we would need to change the dataset identifier in this command to the identifier of the alternative dataset that we wished to use. Further information on the various datasets available within GEE and their respective identifiers is available at the GEE website (<https://earthengine.google.com/>) if required.

```
var collectionS1 = ee.ImageCollection('COPERNICUS/S1_GRD')
```

Currently the command will create an object containing all the Sentinel-1 images available in the GEE archives. We then need to perform a series of filtering operations to select only the images for the data products, geographical area and time periods that we are interested in. Firstly, we specify that we only require the Interferometric Wide Swath data acquisition mode. Sentinel-1 acquires data in multiple data acquisition modes however here we are only interested in the Interferometric Wide Swath mode. We do not go into detail here about the different data acquisition modes and data characteristics of Sentinel-1 but would encourage the user to familiarise themselves with the literature on this topic. The following line specifies that the collection will be filtered to retain only the instrument mode = Interferometric Wide Swath, identified in the code as `IW`.

```
.filter(ee.Filter.eq('instrumentMode', 'IW'))
```

The Sentinel-1 SAR data is acquired in two polarisations, VV and VH. We require both of these polarisations for the subsequent data analysis and here we perform further

filtering steps to retain only the images that contain firstly the VV, then secondly the VH polarisations. We also filter the collection to retain images acquired only in a descending orbital path.

```
.filter(ee.Filter.listContains('transmitterReceiverPolarisation', 'VV'))  
.filter(ee.Filter.listContains('transmitterReceiverPolarisation', 'VH'))  
.filter(ee.Filter.eq('orbitProperties_pass', 'DESCENDING'))
```

Next, we will further filter the collection to retain images intersecting our area of interest. The area of interest here is defined by the object `table`, which is the area of interest bounding box shapefile that was imported as an asset at an earlier stage. This will retain only the images that intersect this area. Note that some of the images retained may not cover the full extent of the area of interest, and the coverage of some images may extend beyond the boundary of the area of interest.

```
.filterBounds(table)
```

Finally, we will filter to retain images acquired only during our time period of interest. To do this we simply specify firstly the start and secondly the end dates of the time period that we are interested in. Here we have specified a time period of the 8th October 2016 to the 8th October 2017. For this command, dates are entered in the format YYYY-MM-DD.

```
.filterDate('2016-10-08', '2017-10-08')
```

The filter commands will run sequentially to produce the image collection for our location, time period and data characteristics of interest. Next, we will take this collection and perform further analysis steps on it.

The image collection will contain a large number of images acquired on different dates. For some applications, however, we do not want many images but instead just a single

image, which represents the ‘typical’ characteristics of the area of interest over the full time period, rather than just at a single snapshot in time. GEE uses ‘Reducers’ to convert a collection of images into a single output image based on parameters that the user defines. We do not explore reducers in depth here, however we will apply a median reducer to the data collection that we have created. This will, for a given location, calculate the median value of all the unmasked pixels within the collection for a pixel location. This median value then forms the pixel value for that location in the output image. This is repeated for all pixels within the extent of the image collection, producing the output raster band.

We generate the median layer using the code below. Here we combine two commands in the same line of code, with the commands applied sequentially in the order that they are written to create a new image named `VVonly_med` containing the median pixel values of the VV polarisation data from the Sentinel-1 collection previously created. The first command selects only the vv polarisation data from the `collectionS1` collection. The second command creates a new image that contains the median vv pixel values. Note that this creates an individual image, rather than a collection of images as we have done earlier.

```
var VVonly_med = collectionS1.select('VV').median();
```

Next, we repeat this process for the VH polarisation.

```
var VHonly_med = collectionS1.select('VH').median();
```

Finally, we calculate the ratio band. We cannot select this in the same way that we did for the VV and VH bands as the ratio band does not yet exist – we need to calculate it. To do this, we need to map the `vh_vv` function that was defined in the functions section of the code over the `collectionS1` collection, and then apply a median reducer.

```
var ratio_med = collectionS1.map(vh_vv).median();
```

The function is mapped here using the command `collectionS1.map(vh_vv)` where `collectionS1` is the collection on which the function will be mapped, `.map()` is the command to map the function, and `vh_vv` is the name of the function that is being mapped. We then apply a median reducer to generate a single median ratio layer as we did earlier for the VV and VH layers.

5.3.3 Topographical data processing

Topographical data is utilised within this analysis in two ways, for inclusion in the land cover classification stage, and also to be analysed as independent variables in relation to mosquito abundance. We are interested in four topographical data products, these being elevation, aspect, slope and Topographic Position Index (TPI). All these products use the Shuttle Radar Topography Mission (SRTM) Digital Elevation Model (DEM) dataset already ingested in GEE. First, we use the SRTM DEM data, here denoted by 'USGS/SRTMGL1_003' to create a new image called `dataset`. Note that as we are just using a single DEM dataset rather than a number of different satellite images as we have done previously, we are creating an image object rather than an image collection.

```
var dataset = ee.Image('USGS/SRTMGL1_003');
```

We then select elevation from `dataset` and use this to create a new `elevation_int` image. We use the `.toInt()` command to convert the image to integer values.

```
var elevation_int = dataset.select('elevation').toInt();
```

From the `elevation_int` image, we then calculate slope and aspect and save them as separate images. Again, both slope and aspect images are converted to integer values, however as converting decimal slope to integer values directly would result in loss of detail within this data, we first multiply the slope values by 10000.

```
var slope10k_int = ee.Terrain.slope(elevation).multiply(10000).toInt();
```

```
var aspect_int = ee.Terrain.aspect(elevation).toInt();
```

Next, we calculate the Topographic Position Index (TPI). To do this, we calculate the elevation for each pixel, and subtract this from the mean elevation of the surrounding area in this case over a 15-pixel radius circular area. This creates the `tpi_15_pixel_int` image, where `elevation` is the DEM specified earlier and `15` is the number of pixels defined as the radius of the circular area over which the mean elevation value will be calculated. We then apply the `.toInt()` command to specify that the dataset generated will be in integer format.

```
var tpi_15_pixel_int = elevation_int.subtract(elevation_int.focal_mean(15)).toInt();
```

5.3.4 Sentinel-2 imagery collection processing

Next, we create a new image collection for the **Sentinel-2 optical images** that will be used in combination with the Sentinel-1 and topographical datasets to perform the land cover classification, and to generate vegetation index data products. The commands below create an image collection by selecting data from the Sentinel-2 surface reflectance data archive (identified in GEE as 'COPERNICUS/S2_SR'), and filters the collection based on the extent of the study area as specified in the asset 'table', and by the date range specified. Sentinel-2 optical data, unlike the Sentinel-1 SAR data, is affected by cloud. This is problematic in many parts of the world including many malaria-endemic regions. Fortunately, the functionality of GEE enables a new composite image to be generated from a series of images that may be partially cloud affected, increasing the area of cloud-free coverage for analysis. Cloud masking of the Sentinel-2 imagery is also performed using the Sentinel-2 Cloud Probability product. First, we specify the imagery and cloud probability data products that we wish to use, and set a maximum cloud probability threshold value, here 65, that we wish to apply.

```
var s2Sr = ee.ImageCollection('COPERNICUS/S2_SR');  
  
var s2Clouds = ee.ImageCollection('COPERNICUS/S2_CLOUD_PROBABILITY');  
  
var MAX_CLOUD_PROBABILITY = 65;
```

We then specify the `.filterBounds` extent and date range that will be applied to filter both datasets.

```
var criteriaS2 = ee.Filter.and(  
  ee.Filter.bounds(table), ee.Filter.date('2018-06-20', '2019-06-24'));
```

We then apply these filtering criteria to both the `s2_SR` and Cloud Probability data that we wish to use. For the `s2_SR` collection, we also map the `maskEdges()` function here.

```
var s2Sr = s2Sr.filter(criteriaS2).map(maskEdges);  
var s2Clouds = s2Clouds.filter(criteriaS2);
```

Next we join the `S2_SR` collection with the Cloud Probability dataset to add the cloud mask, creating the new image collection `s2SrWithCloudMask`.

```
var s2SrWithCloudMask = ee.Join.saveFirst('cloud_mask').apply({  
  primary: s2Sr,  
  secondary: s2Clouds,  
  condition:  
    ee.Filter.equals({leftField: 'system:index', rightField: 'system:index'})  
});
```

Next, we create a new image collection from the `s2SrWithCloudMask` collection by mapping the `maskClouds()` function, and retaining only the spectral bands in those images that we require for the land cover classification. Here, we retain bands 2, 3, 4, 5, 6, 7, 8, 8A, 11 and 12. Bands 1, 9, 10 and 11 are optimised for atmospheric applications, which are not relevant in the context of this work and are therefore disregarded here.


```
var S2_bandsubset =  
ee.ImageCollection(s2SrWithCloudMask).map(maskClouds).select('B2','B3','B4','B5','B6','B7','B8','B8A','B  
11','B12');
```

We then apply a median reducer to the `S2_bandsubset` image collection to create the median image `S2_med_bandsubset`.

```
var S2_med_bandsubset = S2_bandsubset.median();
```

Finally, we combine the Sentinel-2, Sentinel-1 and topographical datasets into a single multi-band image on which we will perform the land cover classification. The new image we will create will be called `data_stack`, with the `S2_med_bandsubset` image forming the first bands. To this, we add the `VVonly_med`, `VHonly_med`, `collection_ratio_run_med`, `elevation`, `slope` and `aspect` bands in that order.

```
var data_stack =  
S2_med_bandsubset.addBands(VVonly_med).addBands(VHonly_med).addBands(ratio_med).addBands(  
elevation_int).addBands(slopex10k_int).addBands(aspect_int);
```

5.3.5 Random forest land cover classification

Now we have created the data stack, we can perform the **land cover classification**. First, we create a new feature collection named `polygons` from the `table2` asset that we imported earlier. This contains a series of polygons corresponding to areas of known land cover types that will be used to train the classifier. Each polygon contains the attribute `'classcode'`, which is an integer code where each value corresponds to a different land cover type (Table 1).

```
var polygons = table2;
```

Next, we extract the pixel values for each band in the data stack for the training polygon locations. In the following commands, `classification_training` is the name of the

object containing the training data that we will create, `data_stack` is the name of the imagery data stack that the pixel values are being extracted from, `polygons` corresponds to the training polygon feature collection, and `scale: 10`, and specifies a spatial resolution of 10m. `tileScale: 16` is a parameter relating to the background processing of the datasets, you do not need to change this.

```
var classification_training = data_stack.sampleRegions({  
  collection: polygons,  
  properties: ['classcode'],  
  scale: 10,  
  tileScale: 16  
});
```

Next, we build the random forest classifier and train it with the training data we have just extracted. The classifier that we create will be called `RF_classifier`, `500` sets the number of trees to be used in the random forest classifier, `classification_training` is the training data, and `classcode` specifies the classes that will be used in the classification. `500` trees are specified here, although if memory limit errors are encountered when running the GEE code, this number can be reduced.

```
var RF_classifier = ee.Classifier.smileRandomForest(500)  
  .train(classification_training, 'classcode');
```

Now that we have trained the classifier, we can classify the data stack.

`land_cover_classification` is the name of the output classification, `data_stack` is the input data stack that will be classified, and `RF_classifier` is the random forest classifier that we have just trained.

```
var land_cover_classification = data_stack.classify(RF_classifier);
```

We can then add the classification to the viewer panel to inspect it, displaying the classification using the colour palette that was defined earlier. `Map.addLayer()` is the command to display an object in the viewer panel. `land_cover_classification` is the object that will be displayed, `palette: palette` specifies the name of the colour palette that we wish to apply, and the `min` and `max` values set the range of data values to be displayed. Here there are eight land cover classes present, with the values (1 to 8) denoting different land cover classes. `Map.centerObject()` instructs the viewer panel to zoom to a bounding box of the specified object, here the `table` extent with the zoom level set to 10.

```
Map.addLayer(land_cover_classification,{palette: palette, min:1,max:8});  
Map.centerObject(table, 10)
```

As well as generating the land cover classification, it is also necessary to perform an accuracy assessment to assess its quality. Random forest classifiers can produce out-of-bag accuracy assessment statistics, although these typically over-inflate the reported accuracy of the classification. An alternative that typically provides more realistic accuracy figures for a classification, is to perform an accuracy assessment using an independent validation dataset. Here, the code for both approaches is presented. First, we will perform an accuracy assessment using the random forest out-of-bag approach. The following commands will generate and return, in this order, an error matrix (often also termed a confusion/correspondence matrix); the overall classification accuracy figure, the Producer's and User's accuracy figures for each individual land cover class and the Kappa coefficient. The `print()` command returns the result of the command to the 'Console' tab. In the accompanying script these commands are commented out but are included so that the user can implement the out-of-bag accuracy assessment if they so wish.

```
var RF_classifier_rf_error_matrix = RF_classifier.confusionMatrix();  
print('RF error matrix: ', RF_classifier_rf_error_matrix);  
var RF_classifier_rf_accuracy = RF_classifier.confusionMatrix().accuracy();  
print('RF accuracy: ', RF_classifier_rf_accuracy);
```

```
var RF_classifier_rf_producers_accuracy = RF_classifier.confusionMatrix().producersAccuracy();  
  print('RF producers accuracy: ',RF_classifier_rf_producers_accuracy);  
var RF_classifier_rf_users_accuracy = RF_classifier.confusionMatrix().consumersAccuracy();  
  print('RF users accuracy: ',RF_classifier_rf_users_accuracy);  
var RF_classifier_kappa = RF_classifier.confusionMatrix().kappa();  
  print('RF kappa: ',RF_classifier_kappa);
```

Next, we will perform the accuracy assessment with the independent dataset, which was previously imported as table3. First, we extract the land cover classes for the independent validation point locations, creating an object called validation_extraction.

```
// Get the values for all pixels in the testing validation dataset.  
var validation_extraction = data_stack.sampleRegions({  
  collection: table3,  
  properties: ['classcode'],  
  scale: 10,  
  tileScale: 16  
});
```

Next, we generate a confusion matrix, overall accuracy, Producer's accuracy, User's accuracy and Kappa coefficient and return these to the 'Console' tab.

```
// Accuracy assessment  
var confusionMatrix = ee.ConfusionMatrix(validation_extraction.classify(RF_classifier)  
  .errorMatrix({  
    actual: 'classcode',  
    predicted: 'classification'  
  }));  
  print('Confusion matrix:', confusionMatrix);
```

```
print('Overall Accuracy:', confusionMatrix.accuracy());  
  
print('Producers Accuracy:', confusionMatrix.producersAccuracy());  
  
print('Users Accuracy:', confusionMatrix.consumersAccuracy());  
  
print('Kappa:', confusionMatrix.kappa());
```

5.3.6 Calculate proportional areas of individual land cover classes using moving window

Next, we will calculate how the **proportional coverage of each land cover class** varies across the study area. To do this requires a two-stage process. First, we take the land cover classification and from this generate binary presence/absence layers for each land cover class individually. Secondly, we use a moving window with a kernel size corresponding to the flight range of *An. gambiae* (846 m) to calculate the proportion of the kernel area occupied by the land cover class in question. The moving window performs this calculation for every pixel in the binary presence/absence layers – for a given pixel it creates a 846 m radius buffer around that pixel, calculates the area where the land cover in question is present, then converts this to a proportional value of the overall buffered area. The moving window then moves to the next pixel and repeats the process, and so on until the calculation has been run on every pixel in the binary presence/absence layer for a given land cover class.

The first stage ‘remaps’ the raster values in the land cover classification (where different integer pixel values correspond to different land cover classes) into binary values where ‘1’ is the land cover class of interest, and all other values (the other land cover classes) are given a value of ‘0’. In the code below, we perform this for the forest land cover class, which has the land cover class code ‘1’. Here, the output binary presence/absence layer that will be generated will be called `class1_only`. In the input `land_cover_classification` eight land cover classes are present, with corresponding class codes 1, 2, 3, 4, 5, 6, 7 and 8. As we are only interested in class 1 (forest), we remap the class values to 1 (retaining forest), then 0, 0, 0, 0, 0, 0, 0 to set the values of all other classes to 0.

```
var class1_only = land_cover_classification.remap([1,2,3,4,5,6,7,8],[1,0,0,0,0,0,0]);
```

If we wished to repeat the process for grassland (land cover class code 2), we would adapt the code as below

```
var class2_only = land_cover_classification.remap([1,2,3,4,5,6,7,8],[0,1,0,0,0,0,0,0]);
```

In the accompanying GEE script this is performed for each land cover class, however for brevity we just present these examples here. We then perform the moving window calculation to generate the proportional coverage of the land cover class in question (here forest, class 1). This produces the output object `class1_mean_mw`, specifies `class1_only` as the input raster, `focal_mean()` is the command to run the moving window, `flight_distance` corresponds to the 846 m flight distance of *An. gambiae* (this was specified earlier in the script), `'circle'` sets a circular (rather than square) kernel to be used, and `'meters'` specifies that the flight distance set is in metres, rather than pixels. Again, in the GEE script this is performed for each land cover class although just an individual example is presented here.

```
var class1_mean_mw = class1_only.focal_mean(flight_distance,'circle','meters');
```

5.3.7 Calculate 'distance to' rasters

Next, we will generate further **raster data** layers giving the distance from a pixel location to the nearest patch of a specific land cover class of interest. This is of particular interest for woodland edges (resting habitat for mosquitos) and water classes (potential breeding habitat). The code below generates a distance product for the forest class, with the output pixel values being distance from the location of that pixel to the nearest woodland patch in metres. In the accompanying GEE script this is also repeated for the fallow, forest and fallow combined, flowing water and static water land cover classes.

```
var dist_to_forest_m_int =  
class1_only.fastDistanceTransform(500).sqrt().multiply(ee.Image.pixelArea().sqrt()).toInt();
```

5.3.8 Vegetation indices

Next, we generate a series of **vegetation index (VI)** and **water index (WI)** data products. For the study area. VI and WI values for the study area will change throughout the year in response to vegetation phenology and seasonal rainfall patterns. Here, we are interested in generating 'typical' VI and WI values across the full year, rather than calculating these values for a particular snapshot of dates within the year. Therefore, we use the Sentinel-2 image collection that we created earlier and use median reducers to calculate median values across the designated time-period. Two VIs, the Normalised Difference Vegetation Index (NDVI) and Soil-Adjusted Vegetation Index (SAVI) will be produced, along with two WIs, the Normalised Difference Water Index (NDWI) and Modified Normalised Difference Water Index (MNDWI). The functions to generate these VI and WI products are pre-defined at the beginning of the GEE script, and here we map these functions over the `S2_bandsubset` image collection. We then calculate the median values for each VI and WI separately.

```
var S2_NDVI_medx10k_int = S2_bandsubset.map(NDVI).median().multiply(10000).toInt();  
var S2_SAVI_medx10k_int = S2_bandsubset.map(SAVI).median().multiply(10000).toInt();  
var S2_NDWI_medx10k_int = S2_bandsubset.map(NDWI).median().multiply(10000).toInt();  
var S2_MNDWI_medx10k_int = S2_bandsubset.map(MNDWI).median().multiply(10000).toInt();
```

After the functions are mapped, we perform a multiplication x10000, then convert the resulting outputs to integer values using `toInt()`. The data is converted to integer format to reduce the size of the data created (integer format data requires less storage capacity than other formats such as float or double). For vegetation index values such as NDVI, which frequently have a value range between -1 and 1, directly converting to integer can truncate the data range resulting in data loss. To prevent this, the data values are multiplied by 10000 before converting to integer, preserving the range of data values. Users should note is subsequently inspecting the data that it has been multiplied by 10000 from the original data values.

5.3.9 CHIRPS data extraction and smoothing

Finally, we will extract **rainfall data** for the study area for the period of, and several months preceding the commencement of the mosquito data collection. Rainfall here is used as a proxy for the availability of mosquito breeding habitat, with rainfall generating ephemeral pools that are frequently used as breeding habitat. The precipitation dataset that will be used is the CHIRPS Daily: Climate Hazards Group InfraRed Precipitation with Station Data (version 2.0 final), identified in GEE as "UCSB-CHG/CHIRPS/DAILY".

The CHIRPS data to be used here - with one CHIRPS dataset for each calendar month - are imported as assets at earlier in the script and do not require any further analysis to within this script. However, should users wish to repeat this processing for alternative sites or dates then the steps to produce these datasets are outlined below. This dataset has a spatial resolution of 0.05 arc degrees and can exhibit a noticeable pixel edge effect. Consequently, we apply a smoothing stage to the CHIRPS data to minimise pixel-edge boundary effects of this dataset. A limit of this analysis is that there is a maximum kernel size limit of 512 pixels that can be used for this smoothing, which at 10 m resolution is an insufficiently small kernel size to perform this smoothing. To work around this limitation, the CHIRPS data can be processed separately and exported to assets at a pixel size of 20 m (rather than 10 m), sufficiently increasing the size of the kernel.

The commands below generate a monthly CHIRPS dataset for the calendar month of August 2014 named `Lodja_mean_2014_aug_smooth`. A series of commands are applied sequentially here. `ee.ImageCollection("UCSB-CHG/CHIRPS/DAILY")` specifies the appropriate GEE dataset identifier from which we will generate a collection. The CHIRPS data is a daily product, but for consistency with the monthly mosquito data collection periods, we will convert this into a monthly mean precipitation per day product. This collection is then filtered by date to retain data from the calendar month of August 2014 using the command `.filter(ee.Filter.date('2014-08-01', '2014-08-31'))` before generating a mean daily value for that time period `.reduce(ee.Reducer.mean())`. The next command `.focal_mean(5000,'circle','meters')`; performs the moving window smoothing, using a 5000 meter radius circular moving window. This is then multiplied by 10000 and converted to integer format.


```
var Lodja_mean_2014_aug_smoothx10k_int = ee.ImageCollection('UCSB-CHG/CHIRPS/DAILY').filter(ee.Filter.date('2014-08-01', '2014-08-31')).reduce(ee.Reducer.mean()).focal_mean(5000,'circle','meters').multiply(10000).toInt();
```

The `Lodja_mean_2014_aug_smoothx10k_int` object will next be exported to assets using the code below. Here, `image:` corresponds to the object to be exported, `description:` is the name of the output file, `scale:` is the spatial resolution of the output file (this is where we specify 20 m), `maxPixels:` sets the maximum number of pixels that can be exported, `region:` sets the extent for which the analysis should be run and data exported (set here to the area of interest defined in the 'table' polygon that was earlier imported as an asset), and `crs:` specified the coordinate system / projection of the output file.

```
Export.image.toAsset({  
  image: Lodja_mean_2014_aug_smoothx10k_int,  
  description: Lodja_mean_2014_aug_smoothx10k_int,  
  scale: 20,  
  maxPixels: 1e13,  
  region: table,  
  crs: "EPSG:4326"  
});
```

This would be repeated for each calendar month required. These datasets will subsequently be read back in as assets, stacked with the other environmental variable datasets generated here, and exported as a stack at 10m resolution so all layers in the stack have a consistent spatial resolution and projection.

5.3.10 Scale and convert data to integer, compile bands and export

We have now generated all the layers of data that we require from GEE. Next, we compile the bands that we require for subsequent analysis and export the data to asset as a multi-band raster, with each band in the stack corresponding to an explanatory variable. It is advantageous to export all the data in this way (as opposed to exporting

each layer individually), as it guarantees that each layer has the same spatial extent, pixel size, number of rows and columns of pixels, and coordinate system.

First, we convert any variables not already in integer format to integer including a multiply by 10000 step to preserve the data ranges where required. This is done for the individual land cover class moving window datasets with an example given below for class 1 (Forest). The user should again be aware that some data layers will therefore be 10000x the original data values in the exported dataset.

```
var class1_mean_mwx10k_int = class1_mean_mw.multiply(10000).toInt();
```

Next, we compile the bands required for export into a new image object called `GEE_data_for_exportx10k_int`. We then display the names of the bands in `GEE_data_for_exportx10k_int` to check that they are all present.

```
var GEE_data_for_exportx10k_int =  
class1_mean_mwx10k_int.addBands(class2_mean_mwx10k_int).addBands(class3_mean_mwx10k_int).a  
ddBands(class4_mean_mwx10k_int).addBands(class5_mean_mwx10k_int).addBands(class6_mean_mwx  
10k_int).addBands(class7_mean_mwx10k_int).addBands(class8_mean_mwx10k_int).addBands(elevatio  
n_int).addBands(aspect_int).addBands(slopex10k_int).addBands(tpi_15_pixel_int).addBands(dist_to_for  
est_m_int).addBands(dist_to_fallow_m_int).addBands(dist_all_forest_and_fallow_classes_m_int).addBa  
nds(dist_to_flowng_water_m_int).addBands(dist_to_static_water_m_int).addBands(S2_NDVI_medx10k  
_int).addBands(S2_SAVI_medx10k_int).addBands(S2_NDWI_medx10k_int).addBands(S2_MNDWI_medx  
10k_int).addBands(Lodja_mean_2014_aug_smoothx10k_int).addBands(Lodja_mean_2014_sept_smo  
othx10k_int).addBands(Lodja_mean_2014_oct_smoothx10k_int).addBands(Lodja_mean_2014_nov_smo  
othx10k_int).addBands(Lodja_mean_2014_dec_smoothx10k_int).addBands(Lodja_mean_2015_jan_smo  
othx10k_int).addBands(Lodja_mean_2015_feb_smoothx10k_int).addBands(Lodja_mean_2015_mar_smo  
othx10k_int).addBands(Lodja_mean_2015_apr_smoothx10k_int).addBands(Lodja_mean_2015_may_sm  
oothx10k_int).addBands(Lodja_mean_2015_june_smoothx10k_int).addBands(Lodja_mean_2015_july_s  
moothx10k_int).addBands(Lodja_mean_2015_aug_smoothx10k_int).addBands(Lodja_mean_2015_sept  
_smoothx10k_int).addBands(Lodja_mean_2015_oct_smoothx10k_int).addBands(Lodja_mean_2015_no  
v_smoothx10k_int).addBands(Lodja_mean_2015_dec_smoothx10k_int).addBands(Lodja_mean_2016_ja  
n_smoothx10k_int).addBands(Lodja_mean_2016_feb_smoothx10k_int).addBands(Lodja_mean_2016_m  
ar_smoothx10k_int).addBands(Lodja_mean_2016_apr_smoothx10k_int).addBands(Lodja_mean_2016_  
may_smoothx10k_int).addBands(Lodja_mean_2016_june_smoothx10k_int).addBands(Lodja_mean_201
```

```
6_july_smoothx10k_int).addBands(Lodja_mean_2016_aug_smoothx10k_int).addBands(Lodja_mean_2016_sept_smoothx10k_int).addBands(Lodja_mean_2016_oct_smoothx10k_int).addBands(Lodja_mean_2016_nov_smoothx10k_int).addBands(Lodja_mean_2016_dec_smoothx10k_int);

print(GEE_data_for_exportx10k_int.getInfo());
```

Next we rename the individual bands with more intuitive names.

```
var GEE_data_for_exportx10k_int = GEE_data_for_exportx10k_int.select(['remapped', 'remapped_1', 'remapped_2', 'remapped_3', 'remapped_4', 'remapped_5', 'remapped_6', 'remapped_7', 'elevation', 'aspect', 'slope', 'elevation_1', 'distance', 'distance_1', 'distance_2', 'distance_3', 'distance_4', 'B8', 'constant', 'B3', 'B3_1', 'precipitation_mean', 'precipitation_mean_1', 'precipitation_mean_2', 'precipitation_mean_3', 'precipitation_mean_4', 'precipitation_mean_5', 'precipitation_mean_6', 'precipitation_mean_7', 'precipitation_mean_8', 'precipitation_mean_9', 'precipitation_mean_10', 'precipitation_mean_11', 'precipitation_mean_12', 'precipitation_mean_13', 'precipitation_mean_14', 'precipitation_mean_15', 'precipitation_mean_16', 'precipitation_mean_17', 'precipitation_mean_18', 'precipitation_mean_19', 'precipitation_mean_20', 'precipitation_mean_21', 'precipitation_mean_22', 'precipitation_mean_23', 'precipitation_mean_24', 'precipitation_mean_25', 'precipitation_mean_26', 'precipitation_mean_27', 'precipitation_mean_28'],

['proportion_forest', 'proportion_grassland', 'proportion_clearing', 'proportion_fallow', 'proportion_built_up', 'proportion_flowng_water', 'proportion_static_water', 'proportion_burnt', 'elevation', 'aspect', 'slope', 'TPI', 'distance_to_forest', 'distance_to_fallow', 'distance_to_forest_or_fallow', 'distance_to_flowng_water', 'distance_to_static_water', 'Median_NDVI', 'Median_SAVI', 'Median_NDWI', 'Median_MNDWI', 'CHIRPS_Aug_2014', 'CHIRPS_Sept_2014', 'CHIRPS_Oct_2014', 'CHIRPS_Nov_2014', 'CHIRPS_Dec_2014', 'CHIRPS_Jan_2015', 'CHIRPS_Feb_2015', 'CHIRPS_Mar_2015', 'CHIRPS_Apr_2015', 'CHIRPS_May_2015', 'CHIRPS_June_2015', 'CHIRPS_July_2015', 'CHIRPS_Aug_2015', 'CHIRPS_Sept_2015', 'CHIRPS_Oct_2015', 'CHIRPS_Nov_2015', 'CHIRPS_Dec_2015', 'CHIRPS_Jan_2016', 'CHIRPS_Feb_2016', 'CHIRPS_Mar_2016', 'CHIRPS_Apr_2016', 'CHIRPS_May_2016', 'CHIRPS_June_2016', 'CHIRPS_July_2016', 'CHIRPS_Aug_2016', 'CHIRPS_Sept_2016', 'CHIRPS_Oct_2016', 'CHIRPS_Nov_2016', 'CHIRPS_Dec_2016']);
```

We then again display the names of the bands in `GEE_data_for_exportx10k_int` to check that they have been changed.

```
print(GEE_data_for_exportx10k_int.getInfo());
```

We then export the data stack to assets:

```
Export.image.toAsset({  
  
  image: GEE_data_for_exportx10k_int,  
  
  description: 'GEE_data_for_exportx10k_int',  
  
  scale: 10,  
  
  maxPixels: 1e13,  
  
  region: table,  
  
  crs: "EPSG:4326"  
  
});
```

5.3.11 Import mosquito sampling data and extract band data

Now that we have created the data stack comprising the explanatory environmental variable bands, we need to extract the values from each band for the time and location at which the mosquito sampling was performed. Firstly, we need to perform a filtering operation to take the full mosquito dataset, and subset it into a series of month-specific datasets. We do this as we do not wish to extract the rainfall values for every month, instead we will only extract the rainfall values for the calendar month in which each subset of mosquito data was collected, and the preceding five months. Consequently, this data must be extracted on a month-by-month basis. The full mosquito dataset has already been imported as an asset named `mosquito_survey_data`. The following code subsets the mosquito dataset, creating a new data subset named `mosquito_survey_data_2015_jan` which contains only the mosquito data collected in January 2015.

```
var mosquito_survey_data_2015_jan =  
mosquito_survey_data.filter(ee.Filter.eq("Year",2015)).filter(ee.Filter.eq("Month",'January'));
```

This performs two filtering operations on the full mosquito dataset, firstly filtering by the year attribute and retaining all records collected in 2015, and then by applying a second filter to retain only the records identified as 'January' in the 'Month' attribute. This process is repeated for each calendar month between January 2015 and December 2016.

Next, we create a series of month-specific data stacks comprising the non-CHIRPS variable bands, and the CHIRPS precipitation bands for the month in question, and the preceding five months. From these, we extract the band values for the mosquito sample locations for the respective month. First, we create a new object named `static_variables` which selects the all the non-CHIRPS bands from the `GEE_data_for_exportx10k_int` data stack.

```
var static_variables = GEE_data_for_exportx10k_int.select(['proportion_forest', 'proportion_grassland',  
'proportion_clearing', 'proportion_fallow', 'proportion_built_up', 'proportion_flowng_water',  
'proportion_static_water', 'proportion_burnt', 'elevation', 'aspect', 'slope', 'TPI', 'distance_to_forest',  
'distance_to_fallow', 'distance_to_forest_or_fallow', 'distance_to_flowng_water',  
'distance_to_static_water', 'Median_NDVI', 'Median_SAVI', 'Median_NDWI', 'Median_MNDWI']);
```

Next, we use the command below to create new data stacks comprising the `static_variables` data stack that we have just created, add to this further CHIRPS precipitation bands corresponding to the month in question and the preceding five months, then apply another command to sample the band values for the locations of mosquito sampling for that given month. The example below is for January 2015, and creates a new object containing the extracted variable data called `training_2015_jan`. A single example is presented here, but in the accompanying GEE script this is performed for all months in 2015 and 2016.

```
var training_2015_jan =  
static_variables.addBands(Lodja_mean_2015_jan_smoothx10k_int).addBands(Lodja_mean_2014_dec_s  
moothx10k_int).addBands(Lodja_mean_2014_nov_smoothx10k_int).addBands(Lodja_mean_2014_oct_  
smoothx10k_int).addBands(Lodja_mean_2014_sept_smoothx10k_int).addBands(Lodja_mean_2014_aug  
_smoothx10k_int).sampleRegions({collection: mosquito_survey_data_2015_jan,properties:  
['An_gambiae'], scale: 10});
```

Once this has been performed for each calendar month, the 24 sets of extracted variable data are then recombined back into a single feature collection object.

```
var training_all_months =  
training_2015_jan.merge(training_2015_feb).merge(training_2015_mar).merge(training_2015_apr).merge(  
training_2015_may).merge(training_2015_june).merge(training_2015_july).merge(training_2015_aug).merge(  
training_2015_sept).merge(training_2015_oct).merge(training_2015_nov).merge(training_2015_dec).merge(  
training_2016_jan).merge(training_2016_feb).merge(training_2016_mar).merge(training_2016_apr).merge(  
training_2016_may).merge(training_2016_june).merge(training_2016_july).merge(training_2016_aug).merge(  
training_2016_sept).merge(training_2016_oct).merge(training_2016_nov).merge(training_2016_dec);
```

We then export this merged dataset so that the next stage of analysis, feature selection using the Boruta method, can be performed in R. The following commands will export the `training_all_months` feature collection to a csv format file named `extracted_data` and save this to the users Google Drive.

```
Export.table.toDrive({  
  collection: training_all_months,  
  description: 'training_all_months',  
  fileFormat: 'CSV'  
});
```

From the users Google Drive this file can then be downloaded, saved locally and processed in R.

5.4. Running the script

This section so far has explained the GEE script content, but not how to run the script. To do this, simply click on the **'Run'** button towards the top-right on the GEE window. This will run the full script except any text that has been commented out. Some commands will write outputs to the **'Console'** tab such as the accuracy assessment results, while export to asset or drive tasks will appear under the **'Tasks'** tab. Any bands or datasets that are added to the map will appear in the viewer at the bottom of the GEE window.

To switch between tabs simply click on the one you require. You can also resize the different panels in the GEE window simply by clicking on the bars between them and dragging. It is particularly useful to expand the map viewer window when inspecting layers that you have displayed. You will need to zoom in on the map to see your area of interest (e.g. Lodja, DRC) (Figure 5.6), this may take a little while to render.

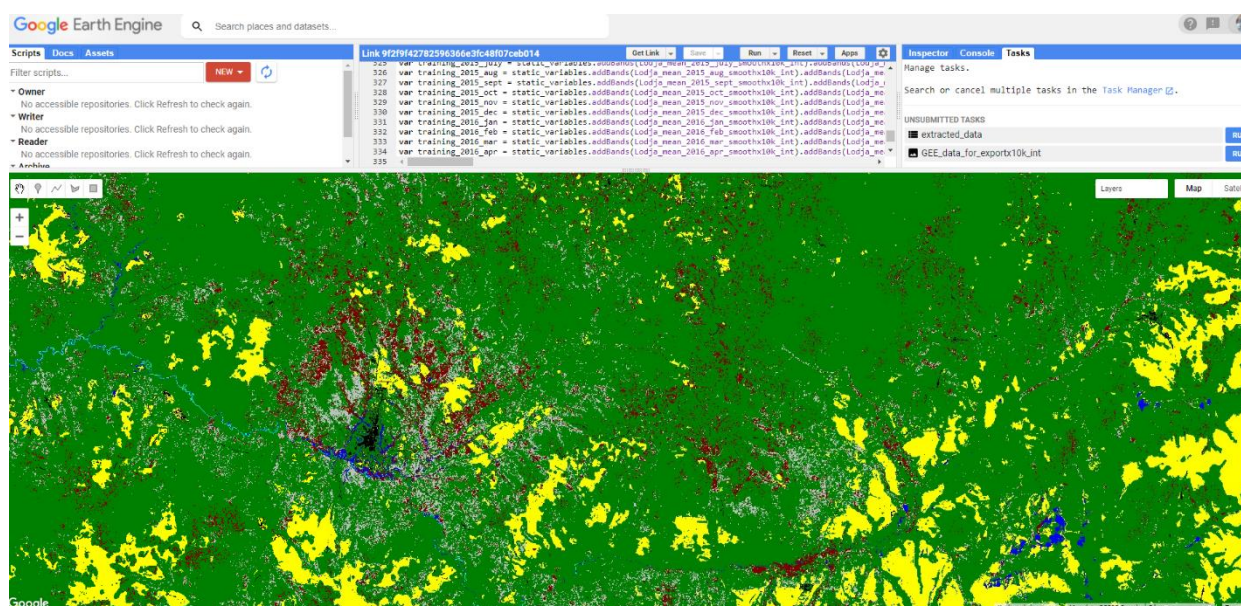


Fig 5.6 Map viewer of Lodja in GEE with land classification shown.

When running export tasks either to asset or drive, as well as running the command to do this in the script, you will also need to click on the **'Tasks'** tab where a list of the

export tasks to be run will be given. There will be a **'Run'** button next to each task – click on this, and the **'Task: Initiate table export'** box will open (Fig 5.7). Click **'Run'** in this window and the export will begin – this is not done automatically, so if you do not click on **'Run'** the export process will not begin. Depending on the size of the object being exported and the volume of processing by other users being run on GEE at the time, exports can take some time – possibly even hours – to complete. Multiple tasks can however be set to run simultaneously, and further processing can also be performed while previous export tasks are still running in the background.

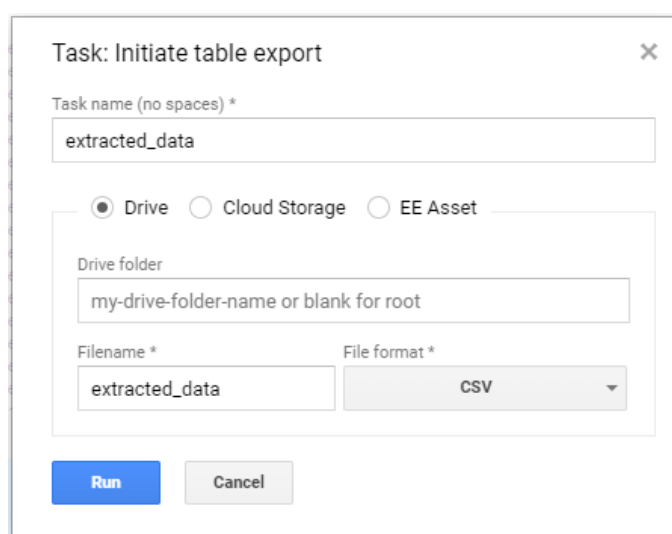


Fig. 5.7 Initiate table export window.

Note that this script involves a large volume of data processing and analysis, so once the script is set running it may take a couple of minutes to complete. You may receive a **'Page Unresponsive'** notification – if you do, do not worry - this simply means the processing is still running. Be patient and give the processing some time to run, the warning box will vanish once the processing has completed.

6. R – feature selection

The next stage of the analysis takes the environmental variable data extracted from the data stack generated in GEE and perform feature selection analysis using the **Boruta method** to identify which of this wider suite of variables are important in relation to *An. gambiae* abundance. As per the GEE section, this stage of the analysis is accompanied by a script that will enable the user to perform the analysis steps in RStudio.

Before beginning the R analysis, we will create a **working directory** in which we will save the data previously extracted and exported from Google Earth Engine. To do this, **open file explorer**, navigate to an appropriate location on your computer and create a new directory by right clicking, selecting '**New**', then '**Folder**'. You can then give the folder that you have created an appropriate name. Although the location and name of the directory you create will vary, for this exercise we will use a directory with the route path: '**D:\R_mosquito_modelling**'. Copy the data to be analysed and the R script that we have provided into this folder. If the location and name of your working directory differs, the R script should be adapted to specify the alternate working directory location and name.

Next, open **RStudio**. You should see a window similar to that shown in Figure 6.1.

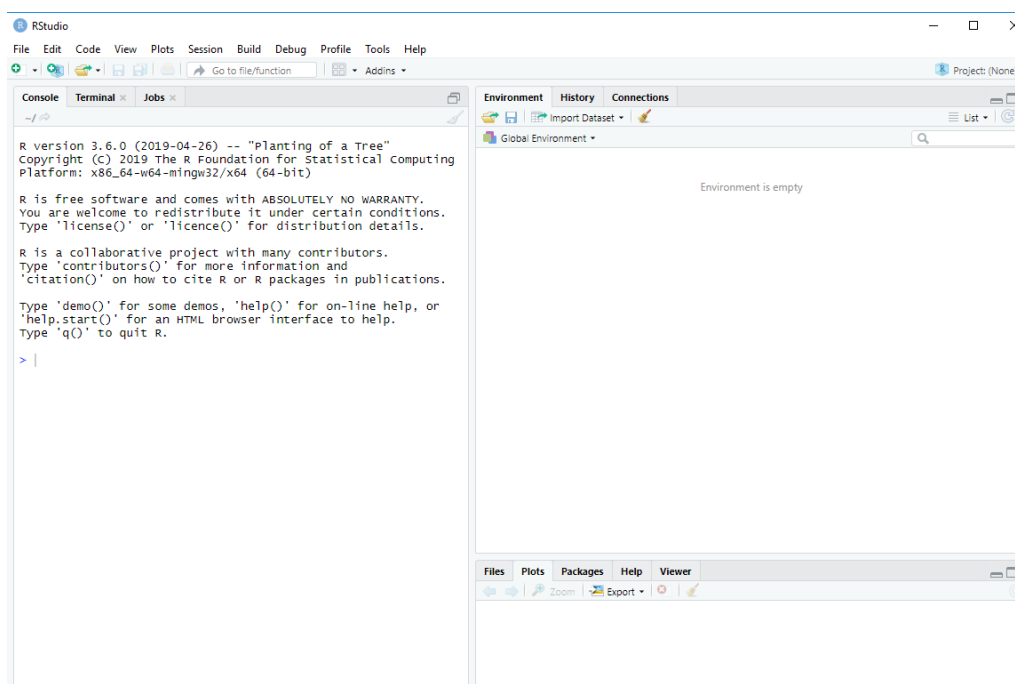


Figure 6.1. The RStudio window

If you are developing a new R script, you would click on **'File'**, then **'New File'**, then **'R Script'**, and an additional window would open in the top left of RStudio where you would write your code. As we have a pre-prepared script available to perform this analysis, we can simply load in this script. Again, go to **'File'**, then **'Open File'**, and navigate to the working directory (**D:\R_mosquito_modelling**). Select the relevant R file containing the script and click **'Open'**. The script should now be displayed in the top left window of RStudio similar to that displayed in Figure 6.2.

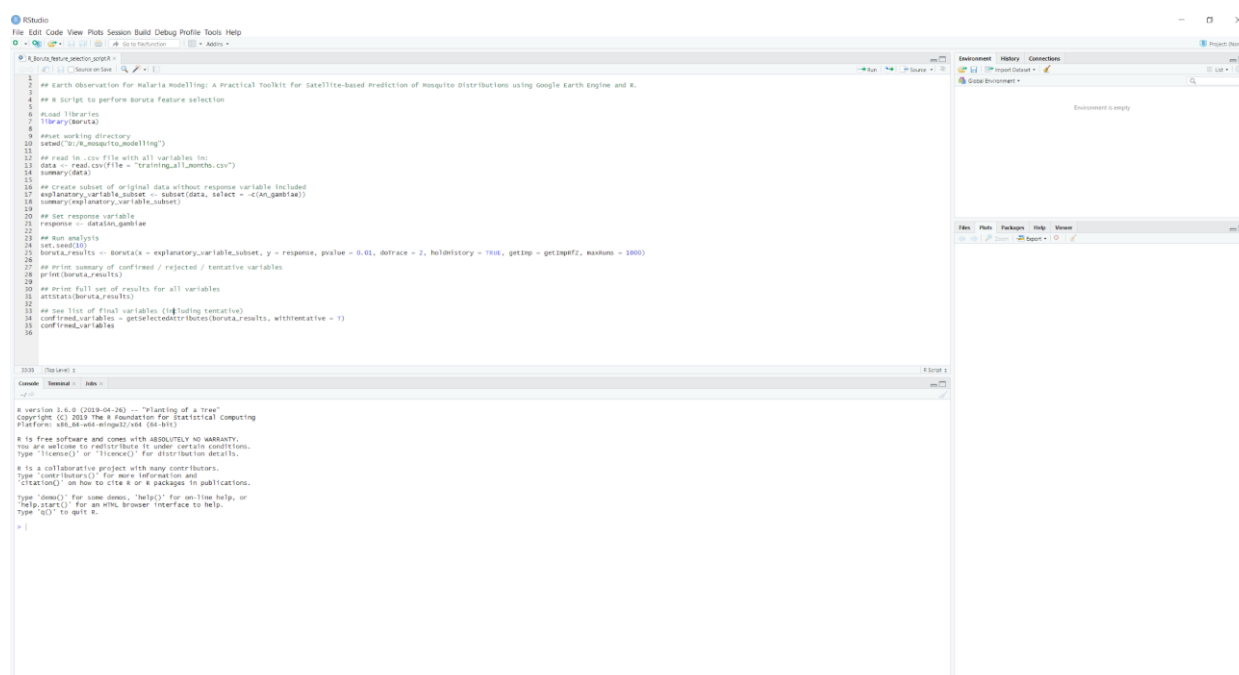



Figure 6.2. R Studio with an example R script loaded.

Similar to in GEE, sections of commands can also be commented out in an RStudio script, with these sections preceded with **#** and usually displayed in green text.

To run the script, or a section of the script, first **highlight the line(s) of code that you wish to run** by clicking the mouse and dragging. Once the correct section(s) of text have been highlighted, click the **'Run'** icon  at the top-right of the window in which the code is displayed.

6.1 Load required packages and set working directory

R contains a wide range of functionality, much of which is available through add-on packages. These packages must be installed and called to enable the functionality of the commands within the package. Packages only need to be installed once but **will need to be called at the beginning of each RStudio session**. Packages can be installed manually via the 'Packages' tab to the bottom right of the RStudio window (Figure 6.3).

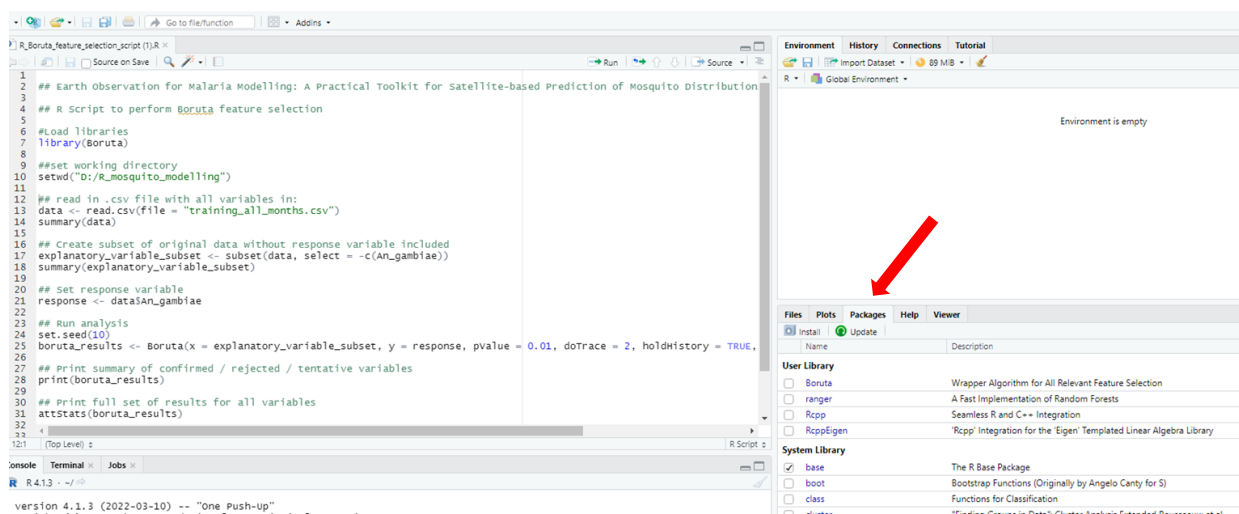


Fig 6.3 Red arrow indicates where to select 'packages'

Click on the 'Packages' tab if this is not already the active tab. Next, click 'Install' and the install packages window should open (Figure 6.4).

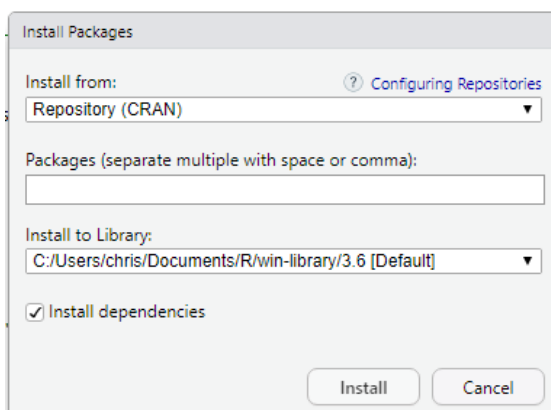


Fig. 6.4. The install packages pop-up window.

In this window, type the name of the package that you wish to install in the '**Packages**' box. As you type, a drop-down list of the available packages should appear. Select the appropriate package name, make sure that '**Install dependencies**' is ticked, then click '**Install**'. **Repeat this process** until all the required packages have been installed. Perform this for the following Boruta package.

We then load the Boruta package using the '**library()**' command. The library command is repeated for each of the packages that are required:

```
#Load libraries  
  
library(Boruta)
```

Next, we specify the working directory created earlier which (unless specified otherwise) R will default to for reading data from and writing data to. If the working directory location and name differs from the one given here, you will need to update the code to reflect this.

```
setwd("D:/R_mosquito_modelling")
```

We then read in the .csv file that contains the mosquito and environmental variable data that we previously exported from GEE. This use the command `read.csv()`, and read in the file named `training_all_months.csv` that is saved in the working directory. As we have already specified the working directory, we do not need to include the full file route path here. This will read in the csv file and create the object `data`. Finally, we produce a summary of the data that we have just imported using the `summary()` command. This will generate basic summary statistics for each of the variables in the `data` dataset that will appear in the console window (Figure 6.4).

```
data <- read.csv(file = "training_all_months.csv")  
  
summary(data)
```

```

> data <- read.csv(file = "training_all_months.csv")
> summary(data)
  An_gambiae      Median_MNDWI      Median_NDVI      Median_NDWI      Median_SAVI      Rainfall.1      Rainfall.2      Rainfall.3      Rainfall.4      Rainfall.5      Rainfall10      TPI      aspect      distance_to_fallow
Min.   : 1.00   Min.   :-5305   Min.   :3234   Min.   :-5423   Min.   :2042   Min.   : 8420   Min.   : 8420   Min.   : 8420   Min.   : 8420   Min.   : 8420   Min.   : 8420   Min.   : -1.00   Min.   : 0.00   Min.   :19.0
1st Qu.: 22.75  1st Qu.: -4971  1st Qu.: 3414  1st Qu.: -4407  1st Qu.: 2272  1st Qu.: 31302  1st Qu.: 31302  1st Qu.: 31302  1st Qu.: 31302  1st Qu.: 32059  1st Qu.: 31302  1st Qu.: 0.00  1st Qu.: 35.75  1st Qu.: 38.0
Median : 38.00  Median : -4334  Median : 3874  Median : -4276  Median : 2387  Median : 48110  Median : 48110  Median : 48110  Median : 48110  Median : 48110  Median : 48110  Median : 2.00  Median : 45.00  Median : 45.0
Mean   : 46.31  Mean   : -4589  Mean   : 3917  Mean   : -4295  Mean   : 2499  Mean   : 42702  Mean   : 43169  Mean   : 42570  Mean   : 42965  Mean   : 43091  Mean   : 43142  Mean   : 1.25  Mean   : 44.25  Mean   : 57.5
3rd Qu.: 63.25  3rd Qu.: -4187  3rd Qu.: 4041  3rd Qu.: -4054  3rd Qu.: 2636  3rd Qu.: 53850  3rd Qu.: 53850  3rd Qu.: 53850  3rd Qu.: 53850  3rd Qu.: 53850  3rd Qu.: 53850  3rd Qu.: 2.00  3rd Qu.: 45.00  3rd Qu.: 89.0
Max.   : 151.00  Max.   : -4059  Max.   : 5338  Max.   : -3605  Max.   : 3371  Max.   : 70638  Max.   : 81707  Max.   : 81707  Max.   : 81707  Max.   : 81707  Max.   : 74025  Max.   : 3.00  Max.   : 153.00  Max.   : 99.0
distance_to_flowin_water distance_to_forest_or_fallow distance_to_static_water elevation proportion_built_up proportion_clearing proportion_fallow proportion_flowin_water proportion_forest proportion_grassland
Min.   : 555.0   Min.   : 0.000   Min.   : 2.000   Min.   : 421.0   Min.   : 1812   Min.   : 113.0   Min.   : 303.0   Min.   : 0.00   Min.   : 5192   Min.   : 93.0
1st Qu.: 586.5   1st Qu.: 0.000   1st Qu.: 3.750   1st Qu.: 425.0   1st Qu.: 1837   1st Qu.: 132.2   1st Qu.: 418.0   1st Qu.: 55.25  1st Qu.: 5389   1st Qu.: 130.0
Median : 639.0   Median : 0.000   Median : 5.000   Median : 423.5  Median : 1876   Median : 142.5  Median : 419.0  Median : 87.00  Median : 5416  Median : 146.5
Mean   : 741.5   Mean   : 3.375   Mean   : 5.375   Mean   : 426.5  Mean   : 1880   Mean   : 138.8  Mean   : 411.1  Mean   : 91.38  Mean   : 5514  Mean   : 143.8
3rd Qu.: 665.2   3rd Qu.: 9.000   3rd Qu.: 7.250   3rd Qu.: 425.0  3rd Qu.: 1910   3rd Qu.: 144.2  3rd Qu.: 423.5  3rd Qu.: 134.75  3rd Qu.: 5472  3rd Qu.: 160.0
Max.   : 1616.0  Max.   : 9.000   Max.   : 9.000   Max.   : 450.0  Max.   : 1984   Max.   : 162.0  Max.   : 460.0  Max.   : 171.00  Max.   : 6324  Max.   : 184.0
proportion_static_water slope
Min.   : 1180   Min.   : 0
1st Qu.: 1861  1st Qu.: 31812
Median : 1879  Median : 39225
Mean   : 1818  Mean   : 33859
3rd Qu.: 1954  3rd Qu.: 43525
Max.   : 1973  Max.   : 52370
  
```

Fig. 6.4. Summary statistics of the attributes of data.

Next, we need to create an explanatory_variable_subset subset data set to retain only the explanatory variables. This uses the subset command, specifying data as the source dataset to subset from, and uses the select = -c(An_gambiae) command to remove the An_gambiae attribute. We then again use the summary() function for the new explanatory_variable_subset object to check that the An_gambiae attribute has been removed.

```

explanatory_variable_subset <- subset(data, select = -c(An_gambiae))
summary(explanatory_variable_subset)
  
```

We then create a new object called response containing only the An_gambiae response variable.

```

response <- data$An_gambiae
  
```

Next, we run the Boruta feature selection analysis. First we set a seed value of 10. We specify that the output results will be written to an object named boruta_results, specify that the Boruta() command will be used, and specify the datasets containing the explanatory and response variable datasets. A number of internal parameters are then specified. Most of these you will not need to change, however there are options of changing the pValue used, and also to number of runs performed in the feature selection analysis. Increasing the number of results can be useful if variables are returned as 'Tentative', meaning that the feature selection is unable to confirm whether a variable is important or unimportant. Here, a maximum number of 1000 runs is specified.

```
set.seed(10)

boruta_results <- Boruta(x = explanatory_variable_subset, y = response, pValue = 0.01, doTrace = 2,
  holdHistory = TRUE, getImp = getImpRfZ, maxRuns = 1000)
```

As the analysis runs, text will be printed to the console window. The process will end either when all variables have either been confirmed or rejected as important, or if the maximum number of specified runs has been reached. When the processing has finished, we can view the results to see whether each explanatory variable has been confirmed as important, unimportant, or is labelled as tentative. We can view a summary of the results (Figure 6.5) using the command below:

```
print(boruta_results)

> ## Print summary of confirmed / rejected / tentative variables
> print(boruta_results)
Boruta performed 999 iterations in 45.42988 secs.
11 attributes confirmed important: distance_to_floating_water, Median_NDWI, proportion_fallow, proportion_floating_water, proportion_forest and 6 more;
11 attributes confirmed unimportant: aspect, distance_to_fallow, distance_to_static_water, elevation, Median_MNDWI and 6 more;
3 tentative attributes left: distance_to_forest_or_fallow, Median_NDVI, Median_SAVI;
> |
```

Fig. 6.5. Boruta feature selection results.

The results summary will be printed to the console window, and state how many iterations were performed and the total run time for these iterations, and how many attributes (variables) were confirmed as important, unimportant, or tentative, and list some of the variables. Note that if there are a larger number of variables falling into these categories then they may not all be listed here, so we may wish to look at the full set of results using the command below.

```
attStats(boruta_results)
```

Finally, if we wish to get a list of the variables that are confirmed as important, we use the commands below. We do have the option of also including variables that are

labelled as tentative using the `withTentative =` command and specifying either T (to include tentative variables) or F (do not include tentative variables).

```
confirmed_variables = getSelectedAttributes(boruta_results, withTentative = T)
```

```
confirmed_variables
```

The Boruta feature selection results demonstrate that for *An. gambiae* abundance in this case the variables in Table 6.1 are confirmed to be important, whereas the remainder of the explanatory variables are confirmed to be unimportant and can therefore be disregarded from further analysis. Having identified this parsimonious set of variables, we return to GEE to continue the modelling stages focussing on only these parsimonious variables.

Table 6.1. The explanatory variables retained after the Boruta feature selection.

Explanatory variable retained
Proportional coverage of forest
Proportional coverage of fallow
Proportional coverage of flowing water
Proportional coverage of static water
Distance to nearest patch of forest or fallow
Distance to nearest patch of flowing water
Median NDVI
Median SAVI
Median NDWI
Rainfall ⁰
Rainfall ⁻¹
Rainfall ⁻²
Rainfall ⁻³
Rainfall ⁻⁴
Rainfall ⁻⁵

This feature selection process is demonstrated here as users may wish to adapt this processing to different mosquito species in different regions, where an alternative

series of environmental variables to those identified here may exert a greater influence on mosquito abundance.

7. Google Earth Engine - modelling

The second GEE script performs the Random Forest modelling on the reduced set of environmental variables that have been confirmed as important in relation to *An. gambiae* abundance by the Boruta feature selection analysis. The GEE script for this stage of the analysis can be accessed via the link below:

<https://code.earthengine.google.com/167b371d166f68830ff02836bb83d7d0>

As before, a number of assets are already imported for this stage of analysis. Again, we import the AOI bounding box delineating our study area, the mosquito survey data, monthly smoothed CHISPS precipitation data from august 2014 to December 2016, and the data stack comprising all the explanatory variable bands (GEE_data_for_exportx10k_int) that we exported to asset at the end of the first GEE script.

Next, a number of parameters for the random forest analysis are set; `ntrees` (the number of trees in the random forest), `MinLeafPopulation` (creates nodes whose training set contains at least this many points), `maxNodes` (maximum number of leaf nodes in each tree - if unspecified no limit is the default), `variablesPerSplit` (the number of variables per split), and `bagFraction` (the fraction of input to bag per tree).

```
var ntrees = 200;

var MinLeafPopulation = 1;

var maxNodes = null; // (no limit)

var variablesPerSplit = null;

var bagFraction = 0.99;
```

We then take the mosquito survey data that has been loaded as an asset, and again subset this data into a series of month-specific datasets as we did previously in the first

GEE script. This is done for each calendar month, with example code given for January 2015 below.

```
var mosquito_survey_data_2015_jan =  
mosquito_survey_data.filter(ee.Filter.eq("Year",2015)).filter(ee.Filter.eq("Month",'January'));
```

Next, we create a stack of the static (non-rainfall) raster bands identified by the feature selection analysis to be important. We did this previously in the first GEE script, however here we will use a reduced set of the static variables, disregarding those that the feature selection analysis confirmed as being unimportant. We name the new data stack `static_variables` and select the bands of interest from the `GEE_data_for_exportx10k_int` data stack that we created in the first GEE script, and imported earlier as an asset.

```
var static_variables =  
GEE_data_for_exportx10k_int.select(['proportion_forest','proportion_fallow','proportion_flowi  
ng_water','proportion_static_water','distance_to_forest_or_fallow','distance_to_flowi  
ng_water','Median_NDVI','  
Median_SAVI','Median_NDWI']);
```

We then create a series of month-specific data stacks comprised of the `static_variables` data stack that we have just created, and the CHIRPS precipitation data bands for the given month, and preceding five months. The precipitation bands have already been imported as assets at the beginning of the script. The example below is for January 2015, however this is repeated for each calendar month between January 2015 and December 2016.

```
var datastack_2015_jan =  
static_variables.addBands(Lodja_mean_2015_jan_smoothx10k_int).addBands(Lodja_mean_2014_dec_s  
moothx10k_int).addBands(Lodja_mean_2014_nov_smoothx10k_int).addBands(Lodja_mean_2014_oct_  
smoothx10k_int).addBands(Lodja_mean_2014_sept_smoothx10k_int).addBands(Lodja_mean_2014_aug_  
smoothx10k_int);
```

Whereas in the first GEE script the similar month-specific data stacks were used to extract the variable values for the mosquito sample locations in a single step, this time

the data stacks serve two purposes. We will extract the mosquito sample locations as before, but we will also predictively apply the random forest models that we generate on these data stacks to produce predicted mosquito abundance maps for the month in questions across the full extent of the study area. The commands above generate the month-specific data stacks that we require to perform this predictive extrapolation. Next, we use the `sampleRegions()` command to extract the values from all the raster bands for the mosquito sample locations. This is again run for each calendar month with the command for January 2015 given below.

```
var training_2015_jan = datastack_2015_jan.sampleRegions({collection:
mosquito_survey_data_2015_jan,properties: ['An_gambiae'], scale: 10});
```

This creates the object `training_2015_jan`, specifies that the `datastack_2015_jan` is the data stack to extract the variable values from, that `mosquito_survey_data_2015_jan` is the mosquito survey dataset for that month for which the sampling locations will be used to extract the data, `properties: ['An_gambiae']` instructs the `An_gambiae` attribute in `mosquito_survey_data_2015_jan` (which is the *An. gambiae* count for the survey location in question) to also be included in the output dataset, and `scale: 10` specifies the spatial resolution to perform the sampling at (here matching the 10 m resolution of the data stack). We then merge the training data extracted for each month into a single object called `training_all_months` using the command below.

```
var training_all_months =
training_2015_jan.merge(training_2015_feb).merge(training_2015_mar).merge(training_2015_apr).merge(
training_2015_may).merge(training_2015_june)
.merge(training_2015_july).merge(training_2015_aug).merge(training_2015_sept).merge(training_2015_
_oct).merge(training_2015_nov).merge(training_2015_dec)
.merge(training_2016_jan).merge(training_2016_feb).merge(training_2016_mar).merge(training_2016_
_apr).merge(training_2016_may).merge(training_2016_june)
.merge(training_2016_july).merge(training_2016_aug).merge(training_2016_sept).merge(training_2016_
_oct).merge(training_2016_nov).merge(training_2016_dec);
```

Next, we identify the band names of the monthly data stacks that are a required parameter of the random forest modelling. We extract the band names using the

command below for the `datastack_2015_jan` `datastack`. The band names should be identical for each monthly data stack, so we only need to do this once here.

```
var bandNames = datastack_2015_jan.bandNames();
```

We then build the random forest model. The commands creates a model named `rf_regression`, which runs the `ee.Classifier.smileRandomForest` command reading in the `ntrees`, `variablesPerSplit`, `MinLeafPopulation`, `bagFraction` and `maxNodes` parameters set earlier in the script, specifies the output mode as regression (rather than classification), sets `training_all_months` as the training dataset to build the random forest regression model on, `An_gambiae` as the response variable, and the `bandNames` as explanatory variable names that we have just extracted from the data stack (above).

```
var rf_regression =  
  ee.Classifier.smileRandomForest(ntrees,variablesPerSplit,MinLeafPopulation,bagFraction,maxNodes).set  
  OutputMode('REGRESSION').train(training_all_months,"An_gambiae",bandNames);
```

This model is then applied predictively on each monthly data stack, producing a single output raster for each month. The example below is for January 2015, however it is repeated for each month changing the input data stack and output product name accordingly. Here, the output name for the predicted raster is `rf_2015_jan_predict`, `datastack_2015_jan` is the input data stack, and `.classify()` is the command to run the `rf_regression` model predictively.

```
var rf_2015_jan_predict = datastack_2015_jan.classify(rf_regression);
```

Finally, we stack the monthly predicted bands into a single data stack named `RF_predicted` and save this to assets. This will then be the input into the final GEE script which will be used to visualise the *An. gambiae* predicted abundance maps.

```
var RF_prediced =  
rf_2015_jan_predict.addBands(rf_2015_feb_predict).addBands(rf_2015_mar_predict).addBands(rf_2015_apr_predict).addBands(rf_2015_may_predict).addBands(rf_2015_june_predict).addBands(rf_2015_july_predict).addBands(rf_2015_aug_predict).addBands(rf_2015_sept_predict).addBands(rf_2015_oct_predict).addBands(rf_2015_nov_predict).addBands(rf_2015_dec_predict).addBands(rf_2016_jan_predict).addBands(rf_2016_feb_predict).addBands(rf_2016_mar_predict).addBands(rf_2016_apr_predict).addBands(rf_2016_may_predict).addBands(rf_2016_june_predict).addBands(rf_2016_july_predict).addBands(rf_2016_aug_predict).addBands(rf_2016_sept_predict).addBands(rf_2016_oct_predict).addBands(rf_2016_nov_predict).addBands(rf_2016_dec_predict);
```

```
Export.image.toAsset({  
  image: RF_prediced,  
  description: 'RF_prediced',  
  scale: 10,  
  maxPixels: 1e13,  
  region: table,  
  crs: "EPSG:4326"  
});
```

8. Google Earth Engine - data visualisation

To visualise the predicted *An. gambiae* data, the third GEE script has been developed to display the monthly datasets and to enable sharing of the data within a GEE app.

The link for this script is below:

<https://code.earthengine.google.com/5cbd6be91c69680e2ca8ec1f4cea0af1>

This script reads in the RF_predicted data stack asset comprising the *An. gambiae* predicted abundances for each calendar month. The first section of code renames the bands to give them more intuitive names, specifying the month and year to which they correspond, and stating whether the bands are the mean predicted abundance (pred) or the standard deviation (stdev) output. The example below presents the code for renaming the predicted abundance band for January 2015, although in the accompanying GEE script this is performed for all months.

```
var RF_predicted = RF_predicted.select(['classification'], ['January_2015_pred']);
```

We then display the predicted abundance for each month, with example code for January 2015 below.

```
Map.addLayer(RF_predicted, {bands: ['January_2015_pred'], min:0, max:100, palette: ['blue', 'green', 'red']}, "An gambiae predicted abundance January 2015", true);
```

Here, the command `Map.addLayer` displays the specified band in the viewer, `RF_predicted` specifies the data stack that contains the data to be displayed, and `['January_2015_pred']` specifies the individual band within `RF_predicted` data stack that we wish to display. `min:0, max:100` specifies the data range to be displayed, and `palette: ['blue', 'green', 'red']` gives the colour palette to be used to display the data. Finally `"An gambiae predicted abundance January 2015"` gives the label to be displayed for that layer, and `true` specifies that the layer will be


displayed automatically. This can also be set to `false`, in which case the band will not be automatically displayed, although this can subsequently be displayed by ticking the check-box for that layer which will appear if you hover the cursor over the Layers tab in the viewer window. It is useful when displaying a large number of bands to initially set this value to `false` as sometimes a large number of bands can take a while to render. Any band that you do wish to display can be switched on using its check-box.

Finally, we use the command below to zoom to and centre the viewer extent to `RF_predicted`.

```
Map.centerObject(RF_predicted);
```

The remainder of the command text in the script corresponds to creating a legend to display alongside the data that we have generated. You will not need to change this code so we do not explore it in detail here.

Running the script will display the results in the map viewer; however ideally, we would be able to disseminate the results to others in a convenient way. GEE allows us to do this via its Apps functionality, which enables others to view the results of the analysis via a web browser.


To publish the script as an app, click on the '**Apps**' button at the top-right of the GEE screen . On the '**Manage Apps**' window that opens, click '**New App**' (top right), and the '**Publish New App**' window will open (Figure 8.1)

Publish New App

App Name ⓘ
My App
The App ID will be generated from the app name. [Edit](#) ▾
URL: https://cmarston_CEH.users.earthengine.app/

Google Cloud Project ⓘ
ee-cmarston [CHANGE](#)

Access Restriction ⓘ
 Restrict access to this app

Public Gallery
 Feature this app in your [Public Apps Gallery](#)
Set Thumbnail (Optional) Description (Optional)
 This app is powered by Google Earth Engine.

Source Code ⓘ
 Current contents of editor
 Repository script path

When the app is published, it's public and anyone can view it. The published source code will be publicly readable. All assets must also be shared publicly or with the app to display properly. See <https://developers.google.com/earth-engine/apps> for more information about publishing apps.

[CANCEL](#) [PUBLISH](#)

Figure 8.1. The Publish New App window.

Give the app an appropriate name, check that under '**Source Code**' current contents of editor is checked, and click '**Publish**'. The App will then appear in the '**Manage Apps**' window. If you subsequently make edits to the script, you will also need to update the App. To do this, in the '**Manage Apps**' window, under the '**ID (click to update app)**' column click on the link for the app and the '**App Details**' window will open. This looks very similar to the '**Publish New App**' window, however under '**Source Code**' at the bottom of this window, check '**Current contents of editor**' then '**Save**' and this will update the App to include the changes made to the source code.

To then launch the App, click on the App name under the '**App Name (click to launch)**' column, and the app should open a web browser and display the data similar to shown in Figure 8.2.

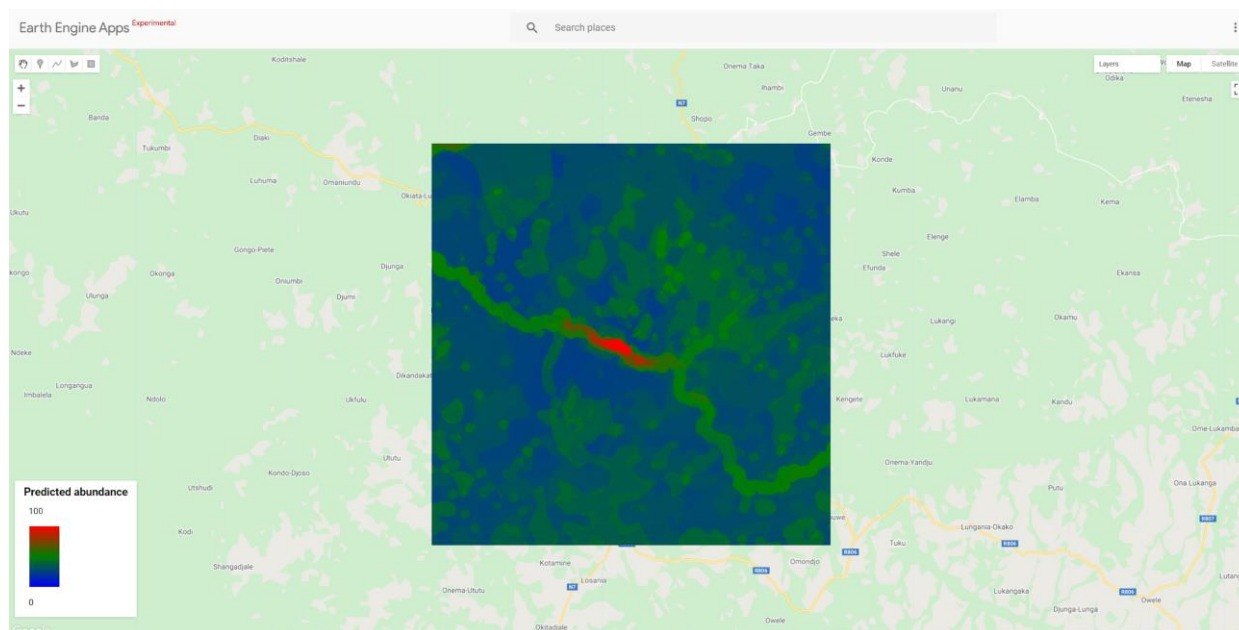


Figure 8.2. Google Earth Engine App displaying the predicted mosquito abundance data.

The App will display the data, but not give the viewer access to the assets or underlying code. By hovering over the **'Layers'** button (top-right) a drop-down list will appear of all the layers available to display. Most of these are currently set not to display, although the user can use the check-boxes for each band to switch on or off the display for each individual band as they wish. They can also zoom in or out and pan around the study area, and also change the base layer from the map displayed in Figure xx to a satellite image layer. Note that the satellite base layer is not the satellite imagery used within the analysis workflow that we have conducted, but is a higher resolution dataset made available within GEE for context / visualisation but not analysis. Note also that there will be temporal offsets between the acquisition dates of the satellite imagery we have analysed, and the acquisition dates of the satellite basemap.

The app link as displayed in the web browser can then be distributed and viewed by users with only the need for a web connection.

Note that to display data in this way the assets that are loaded and displayed (here the RF_predicted stack of bands corresponding to the random forest predictions and standard deviation products for each calendar month must be shared. To do this, return to the GEE code editor containing the script, click on the **'Assets'** tab, and then click again on the asset that you wish to share. This should open a new window similar to the one in UKCEH report ... version 1.0

Figure 8.3.

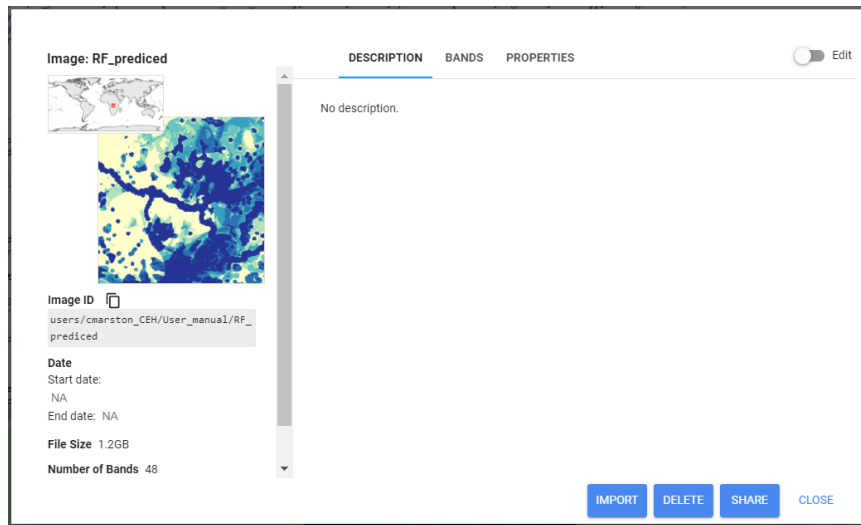


Figure 8.3. Asset information window.

In this window, click on **'Share'**, tick the **'Anyone can read'** check-box, then click **'Done'**. The asset should now be viewable to anyone via the App.

9. Disclaimer

This user guide and sample datasets are provided "as is" without warranty of any kind, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The user assumes all responsibility for the accuracy and suitability of this program for a specific application. In no event will the authors or affiliated institutions be liable for any damages, including lost profits, lost savings, or other incidental or consequential damages arising from the use of or the inability to use this program.

Users are encouraged to engage with these training materials and adapt them for their own purposes. Should publications be developed incorporating the methods presented here, both this user guide and following journal article from which it was developed should be cited.

This user guide should be cited as 'Marston C.G., Rowland C.S., O'Neil A.W., Irish, S., Wat'senga F., Martín-Gallego P., Giraudoux P., and Strode C. 2022. Earth observation for malaria modelling: a practical toolkit for satellite-based prediction of mosquito distributions using Google Earth Engine and R. UK Centre for Ecology and Hydrology, 78pp.'

The journal article should be cited as 'Marston C.G., Rowland C.S., O'Neil A.W., Irish, S., Wat'senga F., Martín-Gallego P., Aplin P., Giraudoux, P. and Strode, C. 2023. Developing the Role of Earth Observation in Spatio-Temporal Mosquito Modelling to Identify Malaria Hot-Spots. Remote Sensing. 15, 43. <https://doi.org/10.3390/rs15010043>'

10. Acknowledgements

This project was funded through a UKRI Collective Fund administered by Edge Hill University as part of the Global Challenges Research Fund, the UK Centre for Ecology and Hydrology (project number NEC07217), and the Natural Environment Research Council award number NE/R016429/1 as part of the UK-SCAPE programme delivering National Capability. Seth Irish was supported by the U.S. President's Malaria Initiative. The findings and conclusions in this paper are those of the authors and do not necessarily represent the official position of the Centers for Disease Control (CDC).

11. References

Marston C.G., Rowland C.S., O'Neil A.W., Irish, S, Wat'senga F., Martín-Gallego P., Aplin P., Giraudoux, P. and Strode, C. (2023). Spatio-temporal modelling of malaria hotspots using Google Earth Engine and quantile regression forests. *Remote Sensing*, 15, 43. <https://doi.org/10.3390/rs15010043>.

Verdonschot, P. F., & Besse-Lototskaya, A. A. (2014). Flight distance of mosquitoes (Culicidae): a metadata analysis to support the management of barrier zones around rewetted and newly constructed wetlands. *Limnologica*, 45, 69-79. doi.org/10.1016/j.limno.2013.11.002

12. Glossary

Land Cover Refers to the surface cover on the ground, whether vegetation, urban infrastructure, water, bare soil or other

Modified Normalised Difference Water Index (MNDWI)

Uses green and short-wave infrared (SWIR) bands for the enhancement of open water features. It also diminishes built-up area features that are often correlated with open water in other indices

Normalised Difference Vegetation Index (NDVI)

Quantifies vegetation by measuring the difference between near-infrared (NIR) (which vegetation strongly reflects) and red light (which vegetation absorbs)

Normalised Difference Water Index (NDWI)

Refers to one of at least two remote sensing-derived indexes related to liquid water; either changes in water content in leaves using NIR and SWIR or changes related to water content in water bodies, using green and NIR wavelengths

Polarisations Polarisation is a way to give transmission signals a specific direction. It makes the beam more concentrated. Signals transmitted by satellite can be polarised in one of four different ways: linear (horizontal or vertical) or circular (left-hand or right-hand)

Rasters A raster consists of a matrix of cells (or pixels) organized into rows and columns (or a grid) where each cell contains a value representing information, such as temperature. Rasters are digital aerial photographs, imagery from satellites, digital pictures, or even scanned maps

Sentinel-1 Sentinel-1A and Sentinel-1B satellites share the same orbital plane. Both use Synthetic Aperture Radar (SAR) has the advantage of operating at wavelengths not impeded by cloud cover or a lack of illumination and can acquire data over a site during day or night time under all weather conditions. Sentinel-1, with its C-SAR instrument, can offer reliable, repeated wide area monitoring. Resolution can be down to 5m and coverage up to 400km

Sentinel-2 Two identical SENTINEL-2 satellites operate simultaneously, phased at 180° to each other, in a sun-synchronous orbit at a mean altitude of 786 km. Sentinal-2 used Visible and Near-Infra-Red (VNIR) bands and Short

Wave Infra-Red (SWIR) bands and monitors variability in land surface conditions under cloud-free conditions

Shapefile A simple, nontopological format for storing the geometric location and attribute information of geographic features. Geographic features in a shapefile can be represented by points, lines, or polygons (areas)

Soil-Adjusted Vegetation Index (SAVI)

A vegetation index that attempts to minimise soil brightness influences using a soil-brightness correction factor. This is often used in arid regions where vegetative cover is low.



BANGOR
UK Centre for Ecology & Hydrology
Environment Centre Wales
Deiniol Road
Bangor
Gwynedd
LL57 2UW
United Kingdom
T: +44 (0)1248 374500
F: +44 (0)1248 362133

LANCASTER
UK Centre for Ecology & Hydrology
Lancaster Environment Centre
Library Avenue
Bailrigg
Lancaster
LA1 4AP
United Kingdom
T: +44 (0)1524 595800
F: +44 (0)1524 61536

EDINBURGH
UK Centre for Ecology & Hydrology
Bush Estate
Penicuik
Midlothian
EH26 0QB
United Kingdom
T: +44 (0)131 4454343
F: +44 (0)131 4453943

WALLINGFORD (Headquarters)
UK Centre for Ecology & Hydrology
Maclean Building
Benson Lane
Crowmarsh Gifford
Wallingford
Oxfordshire
OX10 8BB
United Kingdom
T: +44 (0)1491 838800
F: +44 (0)1491 692424