BBS MAPPING: HOW-TO GUIDE

Details about and how to use the BBS mapping python script which can be found at: **https://github.com/CMcLachlan/BBS_mapping**

CONTENTS

1. BACKGROUND

To carry out breeding bird surveys as part of a standardised monitoring programme for their nature reserves, Essex Wildlife Trust have adapted the British Trust for Ornithology (BTO) breeding bird survey (BBS) methodology, to apply these standardised methods on a smaller scale for monitoring individual nature reserves. Fixed transects are surveyed twice per breeding season, transects are split into approximate 200m sections, and all birds encountered are recorded within a distance of the transect on each side of the surveyor.

Whilst it is valuable to map, visualise and integrate BBS records with other spatial data, manually mapping these records for a large number of transects annually would be extremely time consuming.

1.1. WHAT THE "BBS_MAPPING" CODE DOES

This Python program is intended to enable an automated workflow for efficiently mapping BBS results, given survey data detailing the site, transect section, recording distance and side of transect the record was made on, and a shapefile containing lines representing each transect section on each reserve surveyed.

1.2. HOW TO GET STARTED

1.2.1. Initial set up

Prior to installing and running this program, the user must have set up a GitHub account and installed Git and Conda on their machine. If this has not been completed, use the links below to complete these steps:

- Set up GitHub account to enable use of the repository: https://github.com/
- Git setup: https://git-scm.com/downloads
- Conda setup: https://docs.anaconda.com/free/anaconda/install/
- (Optional) Github desktop: https://desktop.github.com/

1.2.2. Fork and clone the repository

The BBS_mapping repository can be found at the following link: https://github.com/CMcLachlan/BBS_mapping

The steps below outline how to start using the content contained within this repository.

- Find and fork the repository:
  - Follow the link above (https://github.com/CMcLachlan/BBS_mapping) and ensure you are signed in to your GitHub account.
  - Press the "Fork" button (should be visible at the top right).
  - Options for creating the fork will then be displayed. There is a branch under development for mapping polygons representing the BBS recording distance

bands, however this is not required for using the script described here. To only include the main branch in the forked repository, tick “Copy the main branch only”. (If you would like to include all branches, untick “Copy the main branch only”.)

    - Click “create fork”.
    - You should be automatically directed to your newly forked version of the repository.
    - Make a note of the URL, which should be similar to: “https://github.com/<your_username>/BBS_mapping”

- Clone the repository so it can be worked with locally. Instructions below cover how to do this using a command prompt, however Github desktop may alternatively be used if preferred:
    - Open Command Prompt on your computer
    - Type cd followed by the location you want the repository’s files to be stored. For example:

      ```
      cd C:\Users\charl\project
      ```
    - Press enter, then type or paste the following command to clone the repository to your chosen location (replace “your_username” with your GitHub user name):

      ```
      git clone https://github.com/your_username/BBS_mapping.git
      ```

      Press enter. A message should be displayed to indicate that the repository is being cloned and when this has completed.

1.2.3. Set up a conda environment

Set up a conda environment to install the required packages. Instructions below cover how to do this using Anaconda Command Prompt, however Anaconda Navigator or other preferred environment management method may alternatively be used if preferred.

- Open Anaconda Prompt on your computer
- Type cd and type the directory you have cloned the repository to (and press enter to run the command), e.g.:

  ```
  cd c:\Users\charl\project\BBS_mapping
  ```
- Then, use the following command to create the environment using the environment.yml file within the repository:

  ```
  conda env create -f environment.yml
  ```
- After pressing enter, leave the command to run and after some time, a message should appear to confirm that this has completed, as well as how to activate/deactivate the environment.

1.2.4. Create PyCharm project to run and adapt the code

Run the script from within the BBS_mapping environment using your integrated development environment (IDE) of choice (instructions below for setting up PyCharm Community Edition):

- If PyCharm is not already installed, visit https://www.jetbrains.com/pycharm/download/, download the appropriate Community version for the computer you are using, and follow the installation instructions.
- With PyCharm installed, open Pycharm and select "create a new project".
- For "Location", save the project in the same location as the cloned BBS_mapping repository
- To set up a python interpreter, use the conda environment which was created in a previous step. First select "Previously configured interpreter"
- Then press "Add interpreter"
- Select "Conda environment"
- As prompted, provide the path to the conda executable and python interpreter which is part of the previously configured BBS_mapping environment.
    - Conda executable location may vary but should be found in the ~\anaconda3\condabin\conda.bat (or a similar location relative to where Anaconda is installed such as Anaconda3\bin\conda)
    - Python interpreter precise location may vary, but should be found relative to Anaconda in e.g. ~/Anaconda3/envs/BBS_mapping/bin/python.exe or similar depending on operating system.
- Press "OK", then "Create", then "Create from existing sources"
- Double-click "BBS_mapping.py" which should now be available in the side window of the project, and press "run" or shift+F10 to run the script.

1.2.5. Dependencies

The script's main dependencies, detailed in Table 1, are listed in the environment.yml file of the repository for convenient installation of the required packages, excluding 'os' and 'random', which are standard modules in Python and do not require separate installation.

Table 1: Dependencies required to run the BBS_mapping script

Dependency	Purpose
geopandas	Reading and writing files
pandas	Reading and writing files, merging dataframes, returning co-ordinates from a given distance along each record line
shapely	Offsetting transect lines, reading geometry as well-known text, interpolation
os	Listing all files in directory to be processed
random	Generating random numbers between 0 and 1 for plotting points at varying distance along transect section lines, generating random numbers between recording distance band values

1.2.6. Test data

Data is included within the repository folder “data_files/Test”. This includes a transects shapefile, and in “data_files/Test/Records”, a sample of breeding bird survey results relating to the transects included in the shapefile. The headings contained within the test data which are called upon within the script are listed in Table 2. If a different dataset is to be mapped using the script, then the column headings should either match these or be updated within the script by the user if required (the relevant lines of the script are indicated in Table 2 for ease).

Table 2: Column headings included within the test data and called upon within the script (omits survey record columns as these are not directly called upon and do not affect the script's functionality).

Dataset	Heading	Data	Line of script used on
Records	reserve	Reserve name where the survey took place	92, 161
	section	Numbered section of transect where individual record was made	92
	L.R	Contains an L if the record(s) within the row were on the left of the transect during the survey, and an R if the record(s) are from the right side of the transect.	113, 118
	X0.25	Contains an integer representing a number of birds recorded, if encountered between 0 and 25 metres of the transect. Contains NA if no bird was recorded at this distance within the row of data.	50, 53
	X25.100	Contains an integer representing a number of birds recorded, if encountered between 25 and 100 metres of the transect. Contains NA if no bird was recorded at this distance within the row of data.	54, 57
	X100.	Contains an integer representing a number of birds recorded, if encountered over 100 metres away from the transect. Contains NA if no bird was recorded at this distance within the row of data.	58, 61
	flying	Contains an integer representing a number of birds recorded, if encountered flying over the transect. Contains NA if no bird was recorded at this distance within the row of data.	62, 65
Transects	RESERVE	Name of the reserve at which the survey transect is located	92, 144, 163
	SECTION	Numbered section of the transect	93, 144
	Shape_leng	The length of the transect section – not required for the purpose of this program but the heading is used within the script to remove this column from the mapped records.	144
	geometry	Linestring geometry representing the transect section	117, 122, 143, 181

2. METHODS

2.1. Data

A detailed description is provided in Table 2 of the required input parameters for running the script. Other requirements include that transects should be in an appropriate projected coordinate reference system (CRS) designed for accurate measurement of distances. A projected CRS translates positions on the earth from latitude and longitude to Cartesian (x, y) co-ordinates, which can be used by the Shapely package to manipulate geometries (Gilles 2023; Longley *et al.* 2015). For the test data provided, British National Grid is appropriate, as one of many localised projections designed to minimise distortions over a defined area (Longley *et al.* 2015).

2.2. Distance band definition

The data is first manipulated to ensure each survey count of a species is included in a separate row, with the correct distance band defined. The records dataframe is sliced to separate rows with counts included in each of the original distance bands columns in turn. Depending on which column the survey count was located in, the distances in metres from the transect, between which the record should be mapped, are defined. Once columns representing counts from 0-25m, 25-100m, 100m+, and flying over have been manipulated in this manner, the resulting updated records with defined distance bands are merged back together. The slicing method is used, as opposed to iterating through each row and updating in place, because there may be more than one survey count per row (if a species was seen in more than one distance band on the same side, within the same transect section). Therefore, slicing to new temporary dataframes and re-merging afterwards ensures each survey count is assigned a row and distance band.

2.3. Transect geometry joining

A linestring geometry is joined to each row of the records data by merging the records and transects dataframes based on the reserve name and transect section. Pandas "merge" is used for this purpose, and both the reserve name and the transect section must match exactly between each file for a match to be made. This intermediate step provides a linestring geometry for each record based on the appropriate transect section of the transects file.

2.4. Side and distance offsetting

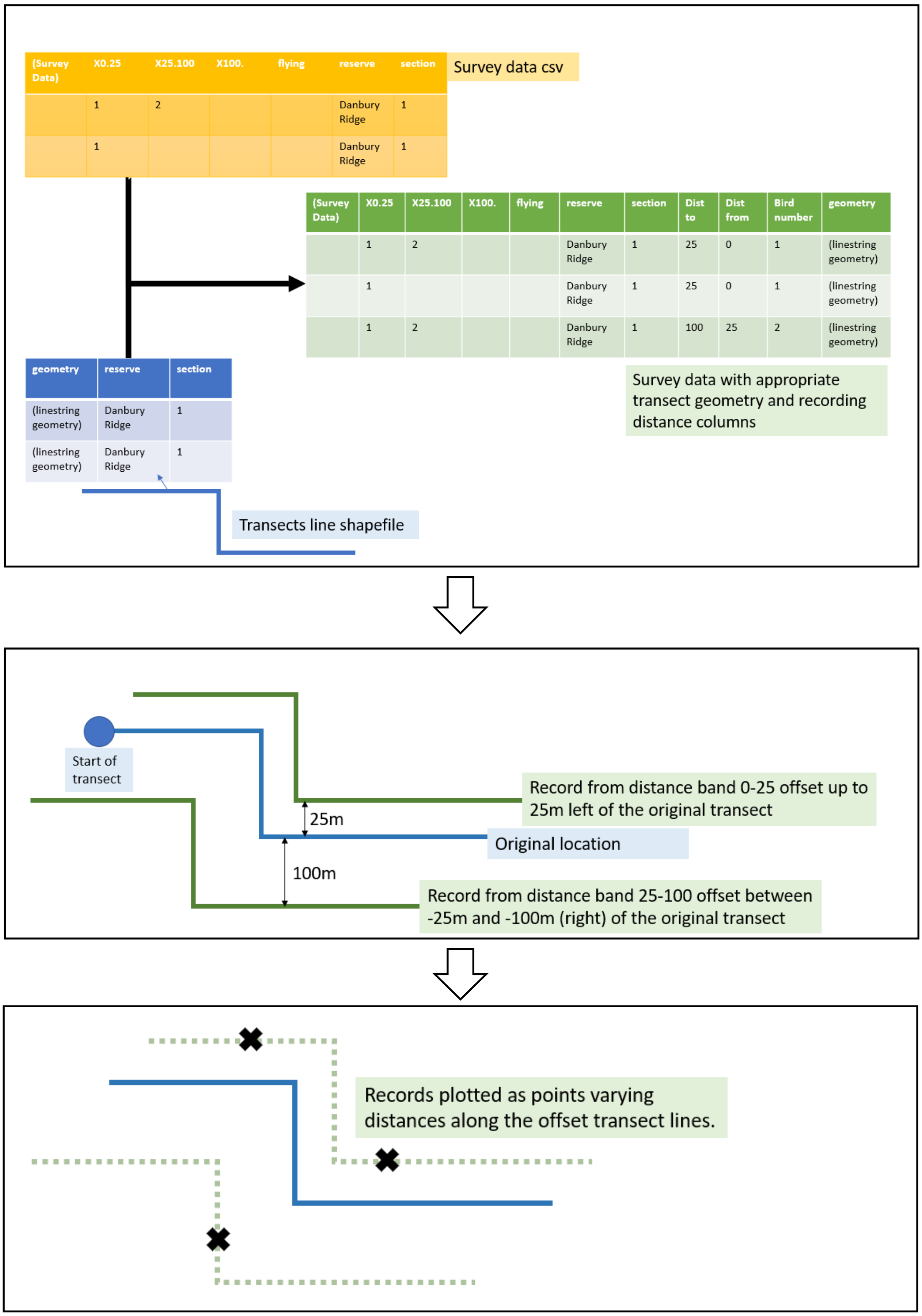
Every linestring geometry is then offset by a distance provided by the initial data manipulation. Rather than offsetting by a specific, prescribed distance, the distances between which the record should be mapped ("distance from" and "distance to") are used as delimiters for generation of a random number. This provides a random distance for each record to be offset by, and improves immediate visualisation of the final mapped data by separating points, making them easier to distinguish visually. For records on the left of the transect, this random number is inserted directly into the Shapely "offset_curve" function. For records on the right of the transect, the random number is first negated, to offset on the opposite (right) side (Gillies 2023).

2.5. Points plotting

A random number between 0.3 and 0.7 is generated to represent a random distance along the linestring geometry of each record. To avoid distortions introduced at the ends of offset lines by the “offset_curve” function, and ensure points are plotted within the correct distance from the transect, the ends of the lines have been avoided by limiting plotting to between 30% and 70% along each line. Interpolation is used to find a point at the random distance along each record’s transect line. The “normalised” argument has been set to “True” so the random number generated is used to represent a proportion of the line’s length (Gillies 2023). The random distance is used for the same reasons detailed in section 2.4.

3. EXPECTED RESULTS

The expected result of running the BBS_mapping code is a set of updated survey records CSV files which contain a geometry column representing a correct point location for each record. The correct location is defined by the recording distance band and side as well as the reserve name and transect number included within the records. A combined point shapefile containing every record input into the script is also created. The points should be plotted at random locations within the appropriate area rather than all points stacked on top of each other at the same location, for ease of viewing the indicative locations of records throughout a reserve prior to layer styling in a geographic information system (GIS). A diagrammatic representation of the expected process of running the code is shown in the figures below. Figure 1 illustrates the key result of each main step taking place throughout the code, and Figure 2 represents an example final point layer which may be obtained.


Survey data csv
Survey data with appropriate transect geometry and recording distance columns
Transects line shapefile
Start of transect
Record from distance band 0-25 offset up to 25m left of the original transect
25m
Original location
100m
Record from distance band 25-100 offset between -25m and -100m (right) of the original transect
Records plotted as points varying distances along the offset transect lines.

(Survey Data)	X0.25	X25.100	X100.	flying	reserve	section
	1	2			Danbury Ridge	1
	1				Danbury Ridge	1

(Survey Data)	X0.25	X25.100	X100.	flying	reserve	section	Dist to	Dist from	Bird number	geometry
	1	2			Danbury Ridge	1	25	0	1	(linestring geometry)
	1				Danbury Ridge	1	25	0	1	(linestring geometry)
	1	2			Danbury Ridge	1	100	25	2	(linestring geometry)

geometry	reserve	section
(linestring geometry)	Danbury Ridge	1
(linestring geometry)	Danbury Ridge	1

Figure 1: Flow diagram illustrating the process of mapping each survey record to the correct location.

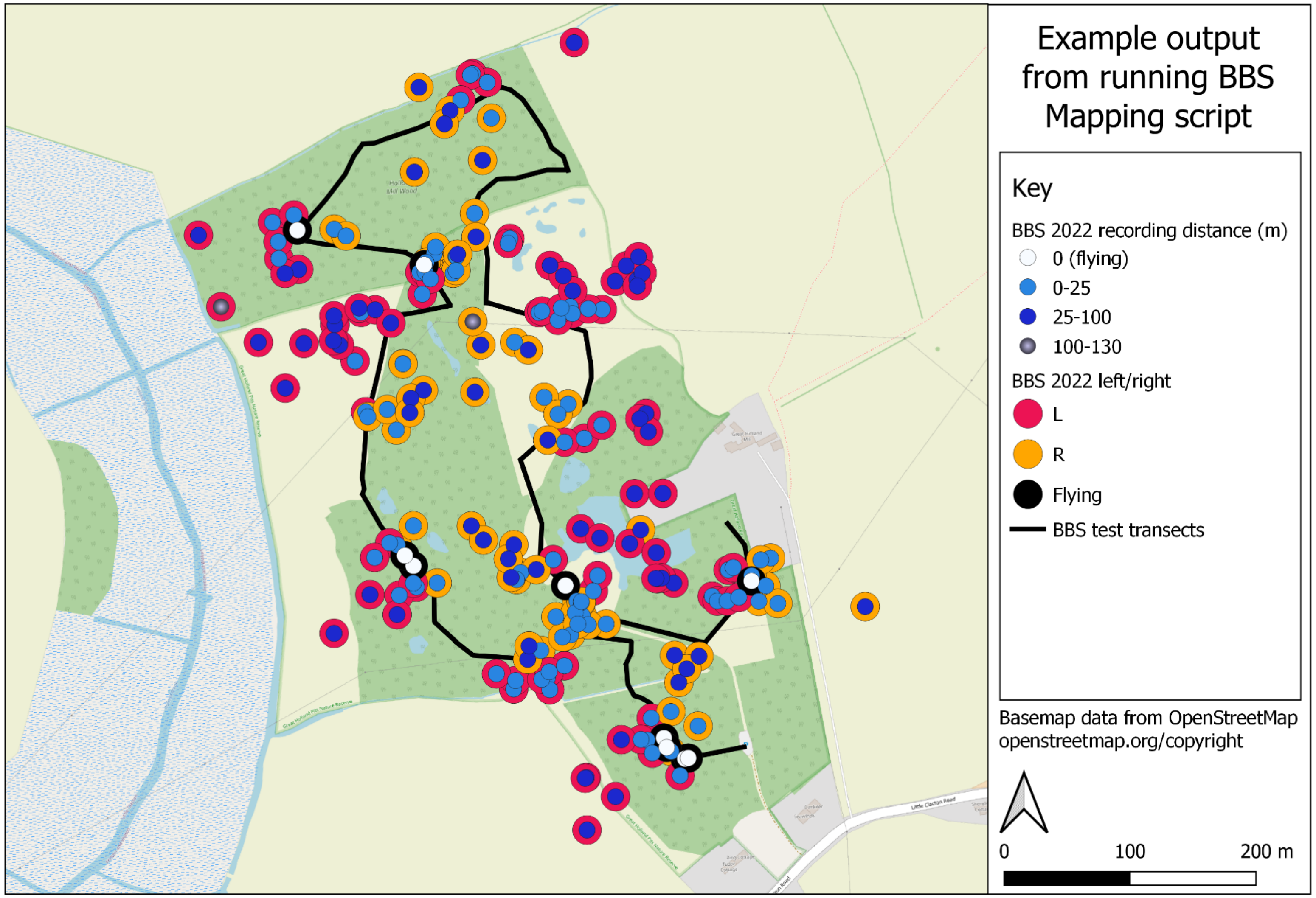

Example output from running BBS Mapping script
Key
BBS 2022 recording distance (m)
0 (flying)
0-25
25-100
100-130
BBS 2022 left/right
L
R
Flying
BBS test transects
Basemap data from OpenStreetMap openstreetmap.org/copyright
0
100
200 m

Figure 2: Map showing an example output for one set of survey records. The point layer has been styled to show both "side" and "distance band" for each record to highlight the information used to map each record to the appropriate location. Each record has been mapped to the correct section, side and distance band relative to the original "BBS test transects" line layer (transect sections not displayed here).

Because of the random number generation within the “read_offset” and “plot_points” functions of the script, it is expected that each time the script is re-run using the same original data, the exact location of each point will change.

4. TROUBLESHOOTING

A) INCORRECT OFFSET DIRECTION: If using any other line/transect data instead of the “BBS_test_transects.shp” provided, ensure the “start” and “end” nodes of the line layer are correct. The left and right sides are determined based on the idea that a surveyor is at the “start” of the transect, facing the “end”. The transect lines must therefore be digitised as such for the offsetting to be carried out on the correct sides.

B) INCORRECT OFFSETTING DISTANCE: A step is included within the code to ensure the transects data used is in an appropriate projected CRS. Ensure that “MyCRS” at the top of the script is set to the appropriate projected CRS for the location. For the test data provided (located in England), “MyCRS” is set to British National Grid.

- If British National Grid is not appropriate for the location of the data being used, update 'epsg:27700' (line 25) to the appropriate European Petroleum Survey Group (EPSG) code.
- If required, to map records from distance bands different to the default values of the script (0-25m, 25-100m, 100m+ or directly on the transect for “flying” records), update the “distfrom” and “distto” values within the “read_dist” function (lines 51, 52, 55, 56, 59, 60, 63, 64). Ensure the distance is provided in metres.

C) PERMISSION DENIED ERROR: Ensure any files being read into the script or overwritten (e.g. previous results if rerunning the script and saving resulting files in the same location) are not open before (re)running the script.

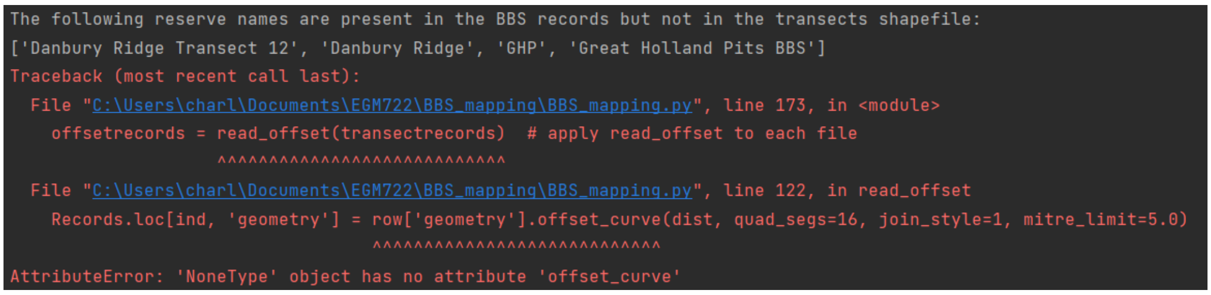
D) COLUMNS NOT FOUND WHEN DROPPING COLUMNS WITHIN “PLOT_POINTS” FUNCTION: Open the transects shapefile within a GIS and check the attribute table for exact column headings which are being joined to each record file.

- Ensure the headings are duplicated exactly in the quote marks in the plot_points function (line 144 - replace the text between the quotes if needed):

```
Records.drop(columns=["Shape_Leng", "RESERVE",
"SECTION"], inplace=True)
```

E) “ATTRIBUTEERROR: 'NONETYPE' OBJECT HAS NO ATTRIBUTE 'OFFSET_CURVE'”: Ensure all records can be successfully joined to a reserve and transect section in the transects shapefile. To do this, ensure reserve names and section numbers for each record in the records CSVs match a reserve name and transect section number in the transects shapefile. There is a section of code within the script which may be used to help with this.

- Uncomment lines 158-167 by removing the triple quotes (”””) from both of these lines, and rerun the script.
- Read the returned text which should appear before the error code as shown in Figure 3. Because this part of the code runs before the error occurs, the set of any reserve names present in the records and not present in the transects file should appear despite the rest of the code not running successfully.
- If any reserve names are listed above the error code, this means there are reserve names present within the survey records which cannot be matched to the reserve name of any of the transects. The data must therefore be amended before the code can run successfully.
    - Open the records file and identify/update any records containing an incorrect reserve name if this was due to a typing error or different abbreviation option being used.
    - Alternatively, open the transects file within a GIS and zoom to the appropriate transect location to make sure the transect has been digitised.
    - If there is a transect present at the expected location, check the attributes of the transect shape at that location and compare the spelling/wording of the reserve name with that in the relevant records file. Ensure that these match before rerunning the script.
    - If there is not a transect present at the expected location, first digitise this (starting at the beginning of the first transect section and digitising all sections in the walked direction), include the correct “RESERVE” and “SECTION” attributes, and save the updated shapefile before rerunning the script.



```
The following reserve names are present in the BBS records but not in the transects shapefile:
['Danbury Ridge Transect 12', 'Danbury Ridge', 'GHP', 'Great Holland Pits BBS']
Traceback (most recent call last):
  File "C:\Users\charl\Documents\EGM722\BBS_mapping\BBS_mapping.py", line 173, in <module>
    offsetrecords = read_offset(transectrecords)  # apply read_offset to each file
                    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\charl\Documents\EGM722\BBS_mapping\BBS_mapping.py", line 122, in read_offset
    Records.loc[ind, 'geometry'] = row['geometry'].offset_curve(dist, quad_segs=16, join_style=1, mitre_limit=5.0)
                                   ^^^^^^^^^^^^^^^^^^^^^^^^^^^^
AttributeError: 'NoneType' object has no attribute 'offset_curve'
```

Figure 3: Example output within PyCharm Community Edition when an error has occurred due to records with the reserve names “Danbury Ridge Transect 12”, “Danbury Ridge”, “GHP” and “Great Holland Pits BBS” being unmatched to a transect geometry due to these names being absent from the transects file. Because of the attribute error which has occurred, lines 158-167 of the script have been uncommented and the code rerun so that the unmatched reserve names are listed before the error.

If, after completing these steps, no reserve names are returned, the transect section numbers should also be checked for correct numbering, and to ensure all section numbers present for each reserve in the records files are present in the transects file. Check the attribute table of the transects shapefile from a GIS and read the records CSV files from an application of choice.

5. REFERENCES

Gillies, S. (2023) *Shapely Documentation.* (Release 2.0.1). Online. Available at: https://buildmedia.readthedocs.org/media/pdf/shapely/stable/shapely.pdf [Accessed 14 May 2023].

Longley, P. A., Goodchild, M. F., Maguire, D. J. and Rhind, D. W. (2015) Geographic Information Science and Systems. Fourth edition. United States of America: Wiley.